FEB V2 Technical Documentation

Draft v0.12

Table of Contents

Change log

March 19th, 2021 -> Document version 0.3

Add a change log, update slow control features, add a lot of raw information, add a Quickstart Guide

July 28th, 2021 -> Document version 0.4 (Firmware revision ID = 1.0)

New BC0 mechanism (signal used as time reference for every other channels)
New downlink frame header FastControl signals for orbit control (FlushDataPath, MuteROCChannels)
Add GBT Rx/TxDataValid flags management
New slow control features (revisionID, PETIROC periodic reconfig and BitflipCounter, Injection Mode)

October 1st, 2021 -> Document version 0.5 (Firmware revision ID = 1.2)

Firmware v1.1 -> Add PETIROC 2A/2B/2C compatibility (modifications of PETIROC Slow Control slave)
Update of the PETIROC configuration registers list
Firmware v1.2 -> Add control of BC0 features (modifications of TDC Slow Control slave)

November 12th, 2021 -> Document version 0.6 (Firmware revision ID = 1.9)

Firmware v1.3 -> Add 10ms delay between each FPGA reset deassertion (to reduce current spikes)
Firmware v1.7 -> Flash interface and Remote update controller implemented (allows to update the application firmware for each of the 3 FPGAs through the GBT frames).
For reliability purpose the flash now contains 2 firmwares (1 golden firmware, 1 application firmware).
Firmware v1.8 -> Add flash burst read management.
Increase delay between FPGAs for reset assertion/deassertion to 50ms.
Firmware v1.9 -> Fix the BC0 timestamp correction when BC0 timestamp is dropped.
Add channel timestamp correction feature.

January 27th, 2021 -> Document version 0.7 (Firmware revision ID = 1.9)

Add details on the Flash interface (to update Application Firmware through the optical link) and the Remote Update Controller (to jump from the Golden Firmware to the Application Firmware).

July 19th, 2022 -> Document version 0.8 (Firmware revision ID = 3.2)

Update PETIROC parameters table to match with CMS database.
Update on the uplink data frame formats (includes new strip clustering features).
Firmware v2.0 -> Add a new Slow Control Slave (Data Path Control registers), introducing a new feature to configure an additional delay for data from Middle FPGA (latency compensation).
Firmware v3.0 -> TDC Data Readout and local Slow Control Buses are now working at 120MHz (clock 100MHz is not used anymore). This implies a modification of the Data Counter Time Window definition (LSB is now 8.33ns instead of 10ns). Add miscellaneous Data Filtering, Strip Clustering and Latency Regulation features.
Firmware v3.1 -> Add an independent pair filtering control for each channel pair (16 enable bits, 16 diff Max and 16 diff Min).
Firmware v3.2 -> Add the possibility to remove isolated channel data on FPGA basis.
Change the Clustering Strip Numbering to better match the PCB strip numbering.
Change the channel/strip readout priority order (channels from even strip first).

November 17th, 2022 -> Document version 0.9 (Firmware revision ID = 3.6)

Change in uplink frame header format (add loopback fields for Resync and BC0 signals in uplink GBT frame).

July 26th, 2023 -> Document version 0.11 (Firmware revision ID = 4.3)

FW v3.7 -> Fix a problem with the FPGA INIT_DONE configuration pin.
FW v3.8 -> Introduce separated reset for each FPGA (using the SCA_Spare(0) GPIOs).
Fix the fast control resetScPath behavior.
FW v3.10 -> Use the internal PLL generated clock as refclk for the GXB module (constant latency).
FW v3.11 -> Optimization of the communication latency between the middle FPGA and the other ones.
FW v3.12 -> Add registers to read the FPGA Chip ID.
Fix the toggling behavior of debug patterns for both data path and sc path (120MHz).

FW v3.14 -> Fix a pipeline synchronization problem in the pair filtering module.
Update the Datapath Control Registers values.
FW v4.0 -> Remove backward compatibility with PETIROC 2A/2B.
Remove PETIROC_CHANNEL_RESET SM in FPGA.
Remove the common soft_reset (SCA GPIO 10).
FPGAs soft_reset are now exclusively independent (SCA GPIO 21, 26, 31).
VAL_EVT of a PETIROC is kept low during ASIC reset or configuration to prevent output toggling.
Remove the features related ROC_MANAGER (registers [4:8] and 13).
FW v4.3 -> Recompile TDC partition for best timing closure (fix the TDC coarse counter problem for the slowest FPGAs).

May 21st, 2024 -> Document version 0.12 (FEBv2r3, Firmware revision ID = 4.8)

Update the documentation to include the FEBv2r3 modifications.
FW v4.4 -> Add a byte ordering detection and correction for links between FPGAs (can correct a problem caused by a SEU).
FW v4.5 -> Register Resync_out, BC0_out and trig_ext output signals in the IO block (this increases the Resync/BC0 to trig_ext latency but reduces the skew between top and bottom injection).
FW v4.6 -> Add Fast-Status flags (header of uplink GBT frame) to tag the loss of data (TDC readout overflow and output frame overflow).
Extend channel dead time counter to 6 bits.
Add a mechanism to mute the PETIROC in case of TDC readout overflow for 1 FPGA.
The val_evt signal is now registered in IO block of the FPGA before to be sent to PETIROC.
FW v4.7 -> Add a PETIROC retriggering mitigation feature (controllable with datapath control registers).
FW v4.8 -> Firmware regeneration to prevent timing issues at worst corner (TDC timestamp observed for one FPGA with the previous version).

Introduction

The goal of this document is to provide information about the Front-End Board V2. This document is currently a draft and modifications/additions could happen as new features are implemented in the firmware.

Overview

FEB environment

This part presents the different devices located on the FEB.

GBTX

This ASIC manages the decoding/encoding of the GBT frames, it ensures the full duplex communication between the backend and the SCA/FPGAs.

Vocabulary:

The transmission path oriented from the backend to the FEB is called the downlink (typically the slow control requests).

The transmission path oriented from the FEB to the backend is called the uplink (typically acquired data and slow control answers).

GBT SCA

This ASIC ensures the board slow control management. It provides I^2C buses, JTAG, SPI interface, GPIOs and ADCs to monitor and control different features of the board.

PETIROC

This ASIC reads the strips and digitizes incoming signal, producing trigger signals to the FPGAs.

In our application, it can be seen as a 1-bit ADC.

There are 6 PETIROCs on the FEB, each of these is managing 16 input channels.

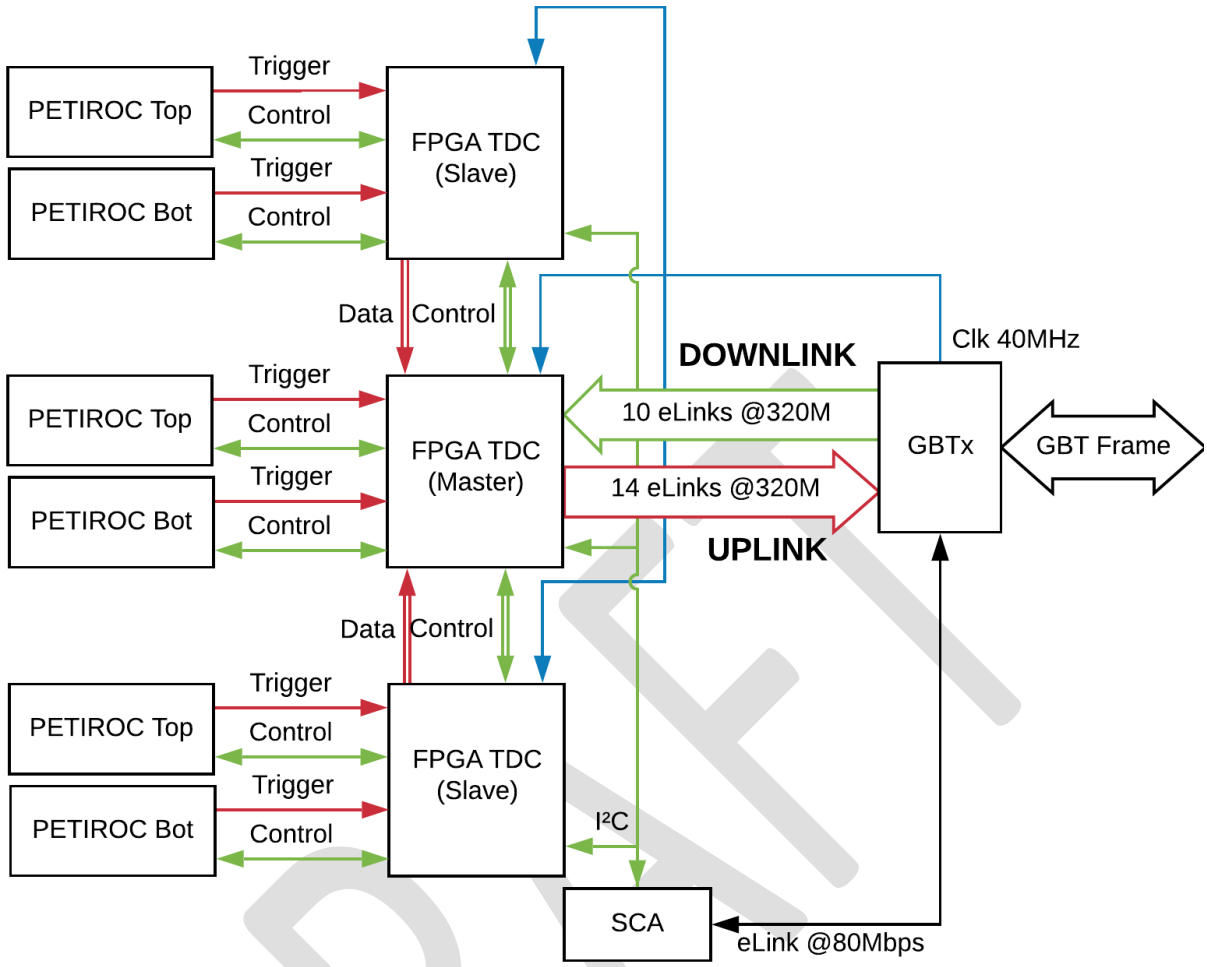
FPGAs

The main task of the three FPGAs of the board is to accurately timestamping the trigger signals received from the PETIROC ASICs. For this purpose, they all three include a multichannel TDC (34 channels).

Each FPGAs has to manage the slow control configuration/monitoring and the fast command generation for its 2 PETIROCs.

In addition, the central FPGA must concentrate data from the two other FPGAs, and format it to create the data payload of the uplink GBT frame.

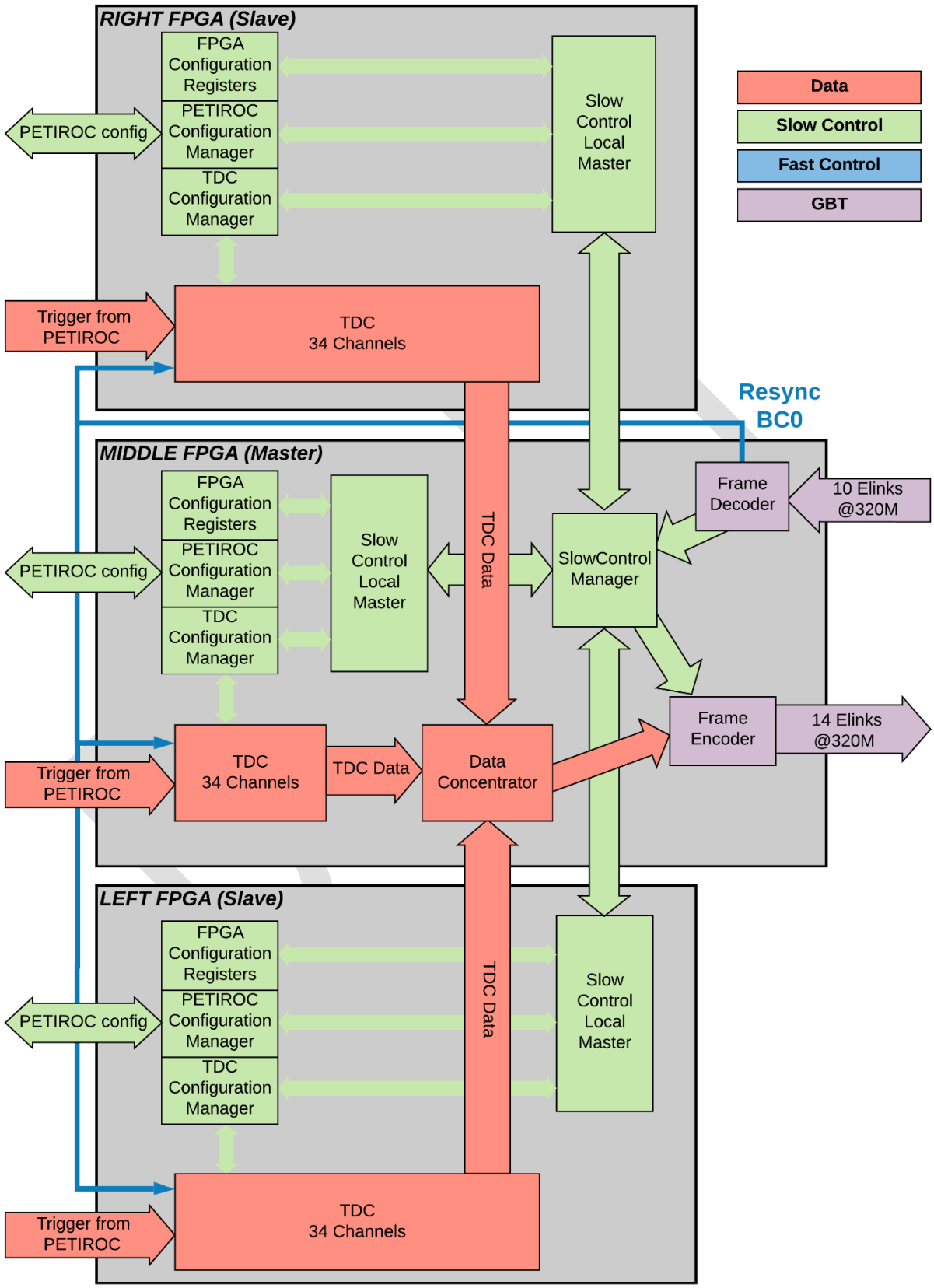
FEB simplified diagram



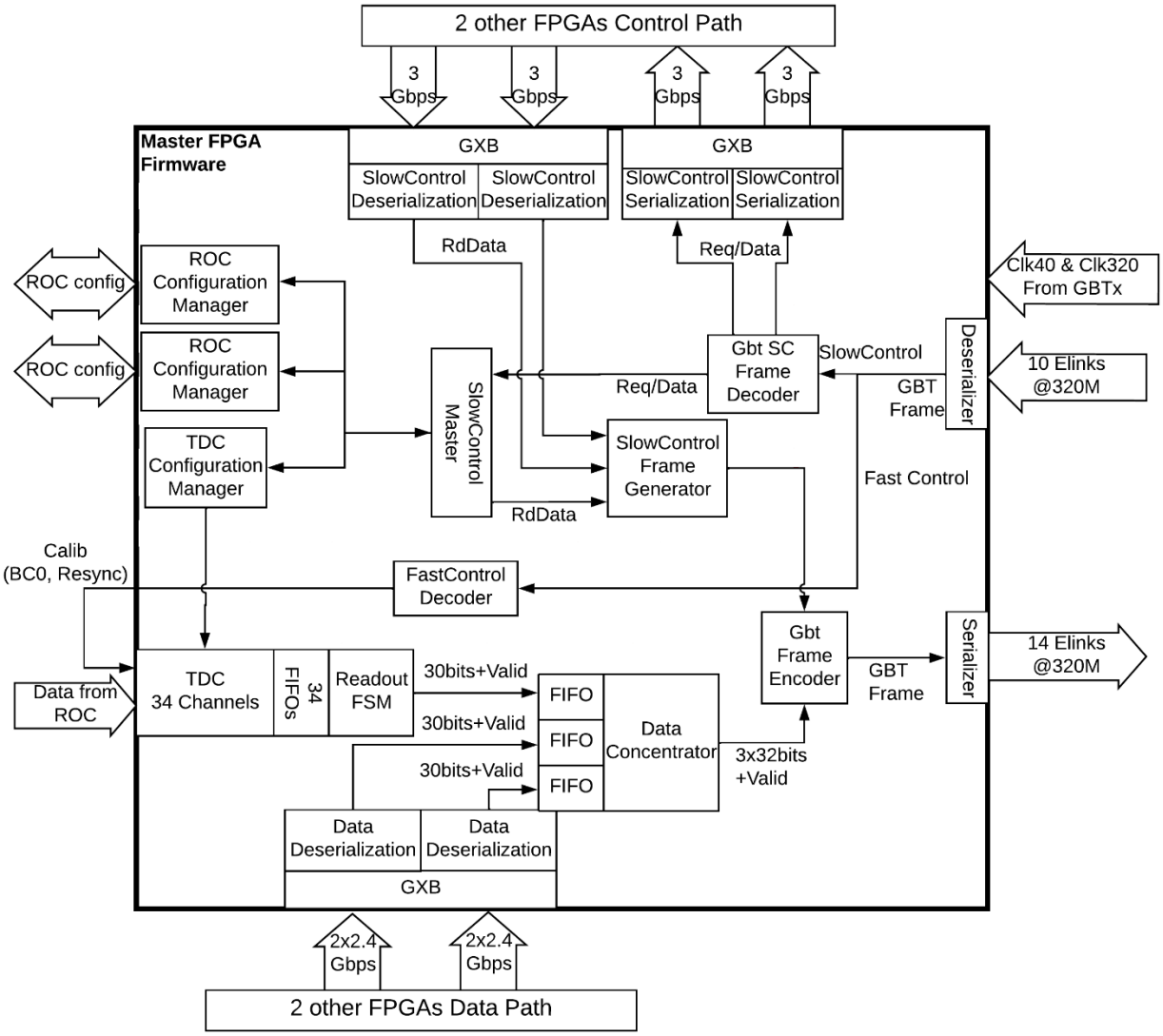
Firmware overview

This part presents how the control and data flows are managed in the firmware of the three FPGAs.

Control and data flows



Focus on the central FPGA



GBT communication

The main communication path between backend and FEB is ensured by a bidirectional optical link. This link data management relies on the CERN GBT solution:

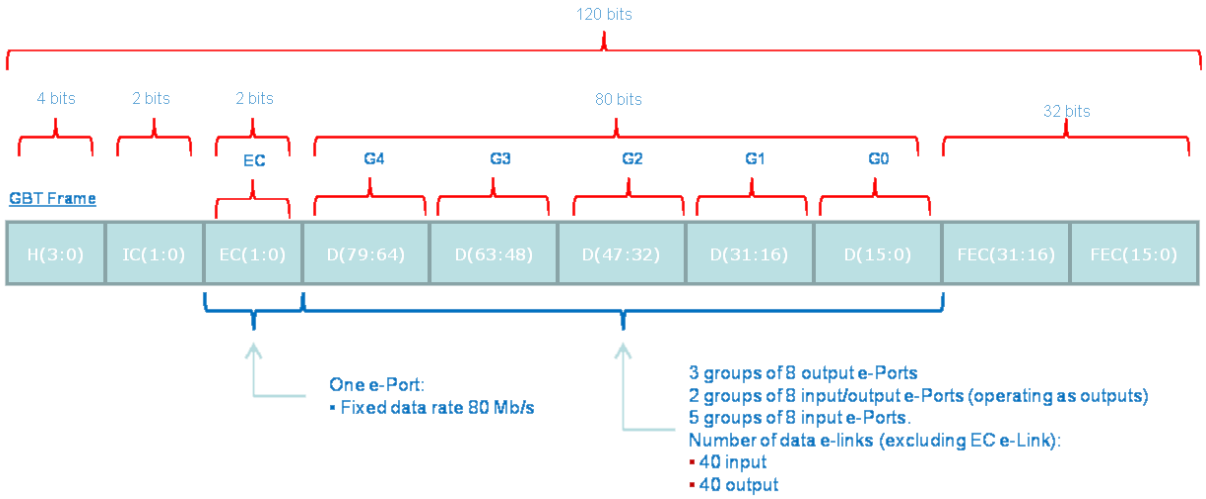
- GBT FPGA on the backend side
- GBTX ASIC on the FEB side

https://espace.cern.ch/GBT-Project/GBTX/Manuals/gbtxManual.pdf

Communication between these two points is based on GBT frames.

GBT frame

The 120 bits GBT frame is transmitted during a single LHC bunch crossing interval (25 ns), resulting in a line rate of 4.8 Gb/s.



Header field (H)

A 4 bits header field is transmitted at the beginning of each frame. This field is required to synchronize the data stream at the frame level. Moreover, once the synchronization is done and the data link is up, this field allows to forward a “data valid” flag from the transmitter side to the receiver side.

Internal Control field (IC)

The 2 bits Internal Control (IC) field is used to control and monitor the GBTX operation. It implements an 80 Mb/s communication link with the GBTX ASIC.

External Control field (EC)

The 2 bits External Control (EC) field is used to implement a fixed bandwidth (80 Mbps) port for the slow control communication channel with the GBT-SCA chip.

Data field (D)

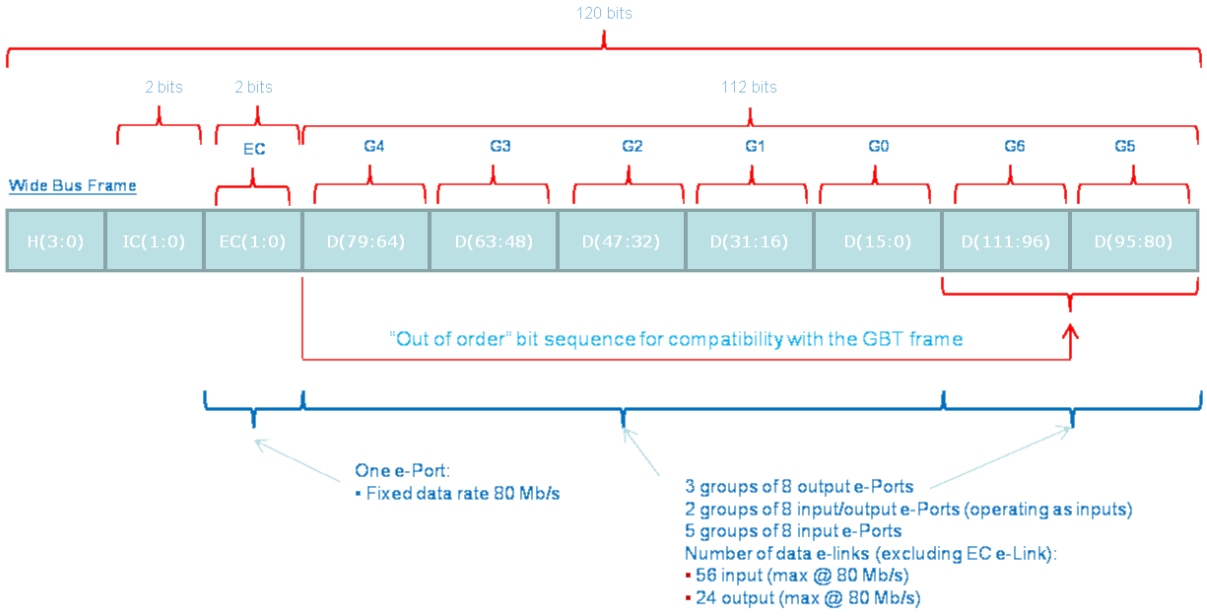
The 80 bits data field is used for generic transmission of data, having an associated bandwidth of 3.2Gbps. On the FEB side, this payload is transmitted/received by the central FPGA via the E-Links. Data transmitted has fixed latency in both directions enabling its efficient use for trigger information and timing control.

Forward Error Correction field (FEC)

The 32 bits FEC field is used to protect all the other fields in the frame against transmission errors due to link noise and single event upsets. Based on this field, the frame receiver can correct up to 16 consecutive corrupted bits.

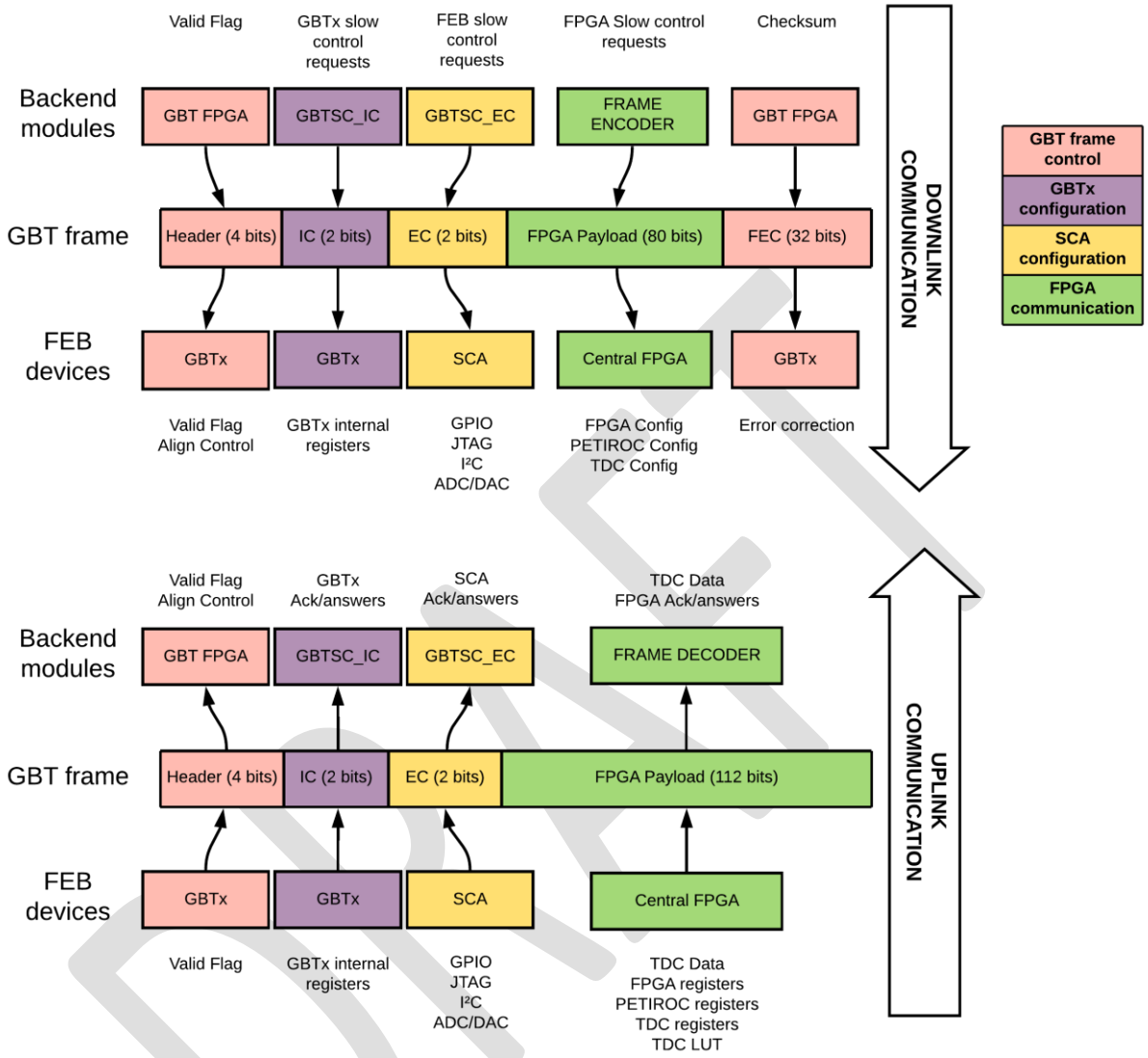
GBT solution allows to increase the uplink useful bandwidth by disabling the Forward Error Correction (FEC) feature and using the corresponding 32 bits field to transmit extra data. Uplink data field therefore has 112 bits, this corresponds to a 4.48 Gb/s useful bandwidth (against to the 3.2 Gbps GBT standard).

It is important to keep in mind there is an asymmetric useful bandwidth between downlink and uplink.



GBT Frame fields source/destination recap

The following figure details the source and destination for each GBT frame field for both downlink and uplink communication.



FPGA communication Frame Format

This communication channel uses the GBT frame data field.

At power-on or after a Reset, the FEB FPGA firmware does not interpret (ignores) all the received frames until the first assertion of the RxDataValid flag provided by the GBTx. This flag is generated by the GBTx based on the GBT frame header which is created by the GBT FPGA (it corresponds to the TxDataValid of the backend board). This is important for the backend to assert this flag before starting the communication with the FEB FPGAs.

Once the FEB is ready to accept and send data AND the RxDataValid flag assertion is detected, the FPGA rises the TxDataValid flag of the GBTx. This can be detected on the backend side by checking the RxDataValid flag status from the GBT FPGA module. This is providing an acknowledge for the backend to confirm the link between the backend FPGA and the FEB middle FPGA is established.

The slow control communication is based on 16bits registers addressed on 16bits.

Downlink GBT Frame

The downlink GBT frame has a useful data payload of 80 bits (5 groups of 16 bits), which is forwarded to the central FPGA.

Downlink communication main goal is to carry 2 types of information: the fast control and the slow control requests.

Slow Control Request Frame	Frame Header (Fast control)							Slow Control Request Information				Slow Control Payload	
Fied Name	Resync	BC0	ResetSCPath	FlushDataPath	MuteROCChannels	MiscCtrl	FPGASel	RSVD	WrReq	BurstAdditionnalWords	Address	WrData0	WrData1
Field length	1	1	1	1	1	8	3	7	1	8	16	16	16
GBT Frame Group	G4							G3			G2	G1	G0

Slow Control Payload Frame	Frame Header (Fast control)							Slow Control Payload			
Fied Name	Resync	BC0	ResetSCPath	FlushDataPath	MuteROCChannels	MiscCtrl	FPGASel	WrData(N)		WrData(N+1)	WrData(N+2)	WrData(N+3)
Field length	1	1	1	1	1	8	3	16		16	16	16
GBT Frame Group	G4							G3		G2	G1	G0

FastControl

A downlink frame has a 16 bits header dedicated to the fast control. These bits are always interpreted and can trigger different actions performed within a fixed latency.

Resync/BC0

These bits both allow to generate a calibration trigger on a dedicated TDC channel for the three FPGAs (through three temporally balanced lines). When a BC0 signal is received and processed by the TDC channel of a FPGA, the associated timestamp is used as a time reference for all channels of this FPGA (for each channel, the returned timestamp value is the difference between the last BC0 reception time and the trigger reception time).

ResetSCPath

This bit is intended to reset the state machines and FIFOs involved in the GBT communication slow control path (on the three FPGAs), this could be useful in case of Req/Data desynchronization (due to an overflow).

FlushDataPath

This bit allows to flush the data pipeline of the three FPGAs of the FEB. Upon reception of this active high control, all the FIFOs and buffers containing TDC data are reset. This feature allows to ensure there is no remaining data at the end of the LHC orbit gap. It removes all ambiguity for the backend during the orbit tagging as this makes impossible to have TDC data from different orbits mixed in the FEB pipeline at the same time.

MuteROCChannels

This bit commands an input dedicated input pin (Val_Evt) of the 6 PETIROC which allows to mute every of their channels (while this bit is active, it is preventing the emission of new triggers from the ASICs). This feature is useful during the orbit gap, when there is a need for masking the generation of non-interesting data while continuing to process and send previously generated data that are already in the FEB FPGAs data pipeline.

MiscCtrl

This is a placeholder for the fast control bits that are not allocated yet.

These bits can be used as synchronous reset for specific part for each of the three FPGAs (TDCs, high speed links connection between the FPGAs, ...), or to synchronize internal counters.

This selection is not fixed yet.

FPGASel

This field allows the firmware to know if any payload (request or data) is present in the frame and to which FPGA(s) it is destined. To transmit the slow control request/data to a given FPGA, the corresponding bit must be set at '1' (this field is not a FPGA address, there is an enable bit for each of the three FPGAs).

Examples:

- "000" -> No slow control forward (empty frame).
- "001" -> Send the slow control payload to FPGA 0 only.
- "110" -> Send the slow control payload to FPGA 1 and FPGA 2.
- "111" -> Broadcast the slow control payload to the three FPGAs.

SlowControl

If at least one bit of the FPGASel field is high, the slow control part of the frame is forwarded to the slow control bus master of the concerned FPGA(s).

A slow control transaction must always begin by a request frame which defines:

- If it is either a read or a write transaction
- If it is either a single word or a burst transaction
- The base address of the transaction

In the case of a write transaction, the value to write is expected to be next to the request information.

In case of a burst write transaction of more than 2 words, the words to write will span over many frames. Unlike the request frame, the following frames have a 4 words maximum capacity.

The slow control path can accept up to 256 words burst request (for both read and write operation).

Note on request/payload frame:

The differentiation between a slow control "request frame" and a slow control "payload frame" is made by the internal state machine which decode the slow control frame in each FPGA:

The very first slow control frame sent to a FPGA is assumed to be a Request one.

If this is a write request, "BurstAdditionnalWords" field is decoded to process the number of remaining frames to get all the necessary data to write (= the number of following frames related to the request).

When all these frames are received, the next frame is assumed to be a Request one.

Example:

You want to write 4 words (0x0A, 0x0B, 0x0C, 0x0D) respectively in registers 0x0010, 0x0011, 0x0012 and 0x0013 in the FPGA 0.

1st Frame (request frame): FPGASel = "001"; WrReq=1, BurstAdditionnalWords=0x03, Address=0x0010, WrData0=0x0A, WrData1=0x0B

2nd Frame (payload frame): FPGASel = "001"; WrData(N)=0x0C, WrData(N+1)=0x0D, WrData(N+2)= XX, WrData(N+3)=XX (The last 32bits of the frame are not interpreted in this case)

After these 2 frames, if another frame is sent to FPGA 0, the frame will be decoded as a request frame (and it will initiate a new transaction).

In this example the 1st Frame and the 2nd Frame don't have to be consecutive.

It is possible to have an arbitrary long break (pause) between the 2 frames as long as the "FPGASel" LSB (which refers to the FPGA 0) stays at '0'.

Note on BurstAdditionnalWords field:

BurstAdditionnalWords = 0x00 -> Single word read or write operation is requested.

BurstAdditionnalWords = 0x01 -> 2 words operation (the single word + 1 additional word)

…

BurstAdditionnalWords = 0xFF -> 256 words operation (the single word + 255 additional words).

This field value is always equivalent as "total number of words" - 1

Uplink GBT Frame

Thanks to the wide frame mode, the uplink GBT frame has a useful data payload of 112 bits (7 groups of 16 bits), which is produced and transmitted by the central FPGA.

Uplink communication main goal is to carry three types of information: the fast status information, the slow control answers to the read requests and the collected data.

The multiplexing of data and slow control frame is based on the very simple rule: "Data always takes priority over slow control" (the output slow control frames are stored in a FIFO which is read only when the frame is free of data).

Frame Header (fast status and valid flags)

An uplink frame has a 16 bits header dedicated to the fast status. These bits allow to monitor the state of the three FPGAs.

Moreover, in this header there are few bits reserved to identify if this is either a data or control (or nothing) in the payload.

	Header (status flags)										
Fied Name	**FC LoopBack**		**Frame Overflow**	**TDC_Readout_Overflow**			**RSVD**	**SCFrame**	**IsStrip**	**RSVD**	**DataValid**
	Resync LoopBack	BC0 LoopBack		FPGA0	FPGA1	FPGA2					
Field length	1	1	1	1	1	1	3	1	2	1	3
GBT Frame Group	G4										

Resync LB/BC0 LB

These two 1bit fields are the loopback of the Resync & BC0 signals extracted from the downlink GBT frame.

Frame overflow

This flag is raised to ‘1’ when at least one data frame is dropped in the uplink GBT frame queue located in the middle FPGA after the data concentrator. An overflow occurring here means that too many timestamps were produced by the 3 FPGAs and therefore the output frame queue hits the maximum authorized latency.

TDC_Readout_Overflow

This flag is raised to ‘1’ when at least one timestamp is dropped at the TDC readout module of one FPGA. An overflow occurring here means that too many timestamps were produced by this FPGA and therefore the TDC readout module hits the maximum authorized latency to send data to the data concentrator.

RSVD

Placeholder for the remaining status bits.

SCFrame/DataValid flags

To differentiate DataFrame and Slow control frame, one has to use the SCFrame flag of the header.

If SCFrame = '0':

Possible data frame, check the 3 “DataValid” bits to know if data is present in frame, and eventually the 2 “IsStrip” bits to know how to correctly reconstruct data.

If SCFrame = '1':

Slow Control frame, check the 6 “DataValid” bits to know where are the slow control reply words to record.

There is no slow control frame structure for the uplink, only words that are sent in the expected order.

Data Frame

If the SCFrame flag in the header is low but at least one data valid is high, there is data in the payload. A data unit composed of both positional (FPGA ID, channel address/strip address) and temporal (TDC generated timestamp) information.

Data Frame in standard mode

The following figure shows the generic data frame format for single channel data.

Data frame	Header (status flags)							TDC Data Standard Payload								
Fied Name	FC LoopBack	Overflow	RSVD	SCFrame	IsStrip	RSVD	DataValid	devAddr	chanID	TDC data	devAddr	chanID	TDC data	devAddr	chanID	TDC data
Field length	2	4	3	1	2	1	3	2	6	24	2	6	24	2	6	24
GBT Frame Group	G4							G3 & G2			G1 & G0			G6 & G5		

In this mode, a given piece of data is 32 bits wide and is related to only 1 TDC generated timestamp.

Data Frame in strip clustering mode

The following figure shows the generic data frame format for strip cluster data.

Data frame	Header (status flags)							TDC Data Strip Payload							
Fied Name	FC LoopBack	Overflow	RSVD	SCFrame	IsStrip	RSVD	DataValid	devAddr	stripID	TDC data	devAddr	stripID	TDC data	Strip0 Diff	Strip1 Diff
Field length	2	4	3	1	2	1	3	2	6	24	2	6	24	16	16
GBT Frame Group	G4							G3 & G2			G1 & G0			G6	G5

In this mode, a given piece of data is 48 bits wide and is the result of a compression of 2 TDC generated timestamps (the pair of channels issued from one physical strip).

The strip data is composed of:

- The direct strip channel data (32 bits)
- The difference between the direct strip channel timestamp and the return strip channel timestamp (16 bits).

Summary of possible data frames

The following figure summarizes every possible frame payload when SCFrame=’0’ (meaning this frame is not a slow control reply).

	Header (status flags)							Frame Data Payload								
Possible Data Frames	FC LoopBack	Overflow	RSVD	SCFrame	IsStrip	RSVD	DataValid									
Empty Frame	Unrelated to Data Frame Payload			"0"	"00"	"0"	"000"	"000000...0000001"								
1 channel data Frame				"0"	"00"	"0"	"100"	devAddr	chanID	TDC data	XXX					
2 channels data Frame				"0"	"00"	"0"	"110"	devAddr	chanID	TDC data	devAddr	chanID	TDC data	XXX		
3 channels data Frame				"0"	"00"	"0"	"111"	devAddr	chanID	TDC data	devAddr	chanID	TDC data	devAddr	chanID	TDC data
1 Strip Frame				"0"	"10"	"0"	"100"	devAddr	stripID	TDC data	XXX			Strip0 Diff	XXX	
2 Strips Frame				"0"	"11"	"0"	"110"	devAddr	stripID	TDC data	devAddr	stripID	TDC data	Strip0 Diff	Strip1 Diff	
Hybrid Frame				"0"	"10"	"0"	"110"	devAddr	stripID	TDC data	devAddr	chanID	TDC data	Strip0 Diff	XXX	
Hybrid Frame				"0"	"01"	"0"	"110"	devAddr	chanID	TDC data	devAddr	stripID	TDC data	XXX	Strip1 Diff	
GBT Frame Group	G4							G3 & G2			G1 & G0			G6	G5	

Note that Strip Frames and Hybrid Frames can only appears when the Strip Clustering feature is enabled in the Middle FPGA (see the Data Path Control section for more information).

SlowControl Frame

If the SCFrame flag in the header is high and at least one data valid is high, there is a read request answer in the payload.

Slow control frame		Header (status flags)					Slow Control Data Payload					
Source FPGA							FPGA0		FPGA1		FPGA2	
Fied Name		FC LoopBack	Overflow	RSVD	SCFrame	DataValid	RdData(N)	RdData(N+1)	RdData(N)	RdData(N+1)	RdData(N)	RdData(N+1)
Field length		2	4	3	1	6	16	16	16	16	16	16
GBT Frame Group		G4					G3	G2	G1	G0	G6	G5

As the slow control answer and the data from the three FPGAs are not concentrated in the same way, this frame format is slightly different. Each FPGA has a reserved field of 2 words to send the slow control read request answers.

TDC module

The main goal of this system is to perform time tagging of the trigger signal coming from the PETIROC. To accomplish this task, each FPGA firmware hosts a TDC module with 34 independent channels. 32 of these channels are connected to PETIROC output triggers and the 2 remaining channels are connected to the Resync and BC0 loopbacks (signals generated by the master FPGA upon reception of a fast control command).

Time reference

When a BC0 signal (provided by FastControl) is received and processed by the TDC channel of a FPGA, the associated timestamp is used as a time reference for all channels of this FPGA (for each channel, the returned timestamp value is the difference between the last BC0 reception time and the trigger reception time).

Data processing

The synchronous time tagging part of this module is working with a 400MHz clock (derived from the 40MHz clock through an internal pll).

When a trigger rising edge is detected for a channel, the module records the value of an internal counter (counting at 400MHz) which gives the coarse time (2.5ns step). Thanks to a second part of the circuit based on a delay chain, this module can also process a fine time on 8 bits. This fine time is corresponding to the fractional part of the coarse time step, this explains how the system can provide TDC data with a LSB around 10ps (=2.5ns/256).

Once the calculation is over, the created timestamp is corrected by the time reference (the last BC0 timestamp is subtracted from the channel timestamp) and is forwarded to the readout module, which goals is to concentrate TDC data from every channel.

Calibration

Before utilization, TDC channels have to be correctly calibrated. It consists of the creation of a LUT (Look Up Table) for each channel. These LUTs store information about the non-linearity corrections of their delay line. The LUT building is done by the TDC itself (corrections cannot be written from the outside) using a dedicated calibration clock. Calibration control/status for all channels are accessible by the slow control registers of the TDC Control slave.

While a calibration procedure is automatically performed inside the FPGAs at start-up (right after the firmware configuration step), it can be useful to manually relaunch the calibration procedure once the board is hot to achieve better performances.

FastControl and orbit management

The orbit sequencing, numbering and the data orbit tagging must be managed by the backend system.

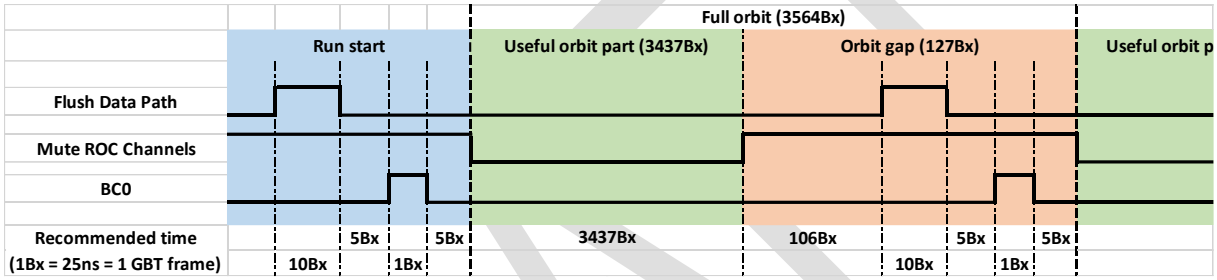
A LHC orbit can be divided in 2 parts:

- The useful part where the input data must be recorded and processed
- The orbit gap where the input data can be dropped

This orbit gap can then be used by the backend to split the received data into different orbits (orbit tagging) with no ambiguity.

To achieve this task, some FEB features can be controlled by the backend using the FastControl bits of the downlink frame header. The effects of these controls have a constant deterministic latency.

The following chronogram is a proposal of how the backend can perform the orbit sequencing by commanding the FlushDataPath, MuteROCChannels and BC0 FastControl bits of the downlink frame.



During the useful part, the 3 controls remain at low level, the FEB continuously generate, process and send data to the backend board.

As soon as the orbit gap begins, the backend board can mute the PETIROCs, preventing them to generate new data. The already collected data processing and sending continue, progressively decreasing the amount of data stored in the buffers.

Near the end of the orbit gap, a flush command needs to be asserted in case of remaining data in the FEB pipeline (this allows to remove every ambiguity as it makes impossible for 2 data from different orbits to be in the FEB at the same time).

After a pause, a BC0 signal can be send to reset the base timestamp of every TDC channels (for the next orbit, all the data timestamps will refer to this time).

The PETIROC can be then unmuted, re-enabling the data acquisition for the next orbit.

FPGA Slow Control structure

The internal Slow Control bus of the firmware is based on 16bits registers addressed on 16bits.

The most significant byte of the address is the slave base address while the less significant byte is the local register address.

Slaves connected to the FPGA slow control bus:

Slave ID	Base address	Slave Name	Access
0	0x0000	FPGA General Control	RW + RO
1	0x0100	PETIROC TOP Control	RW + RO
2	0x0200	PETIROC BOT Control	RW + RO
3	0x0300	TDC Control	RW + RO
4	0x0400	TDC 0 LUT	RO
5	0x0500	TDC 1 LUT	RO
…	…	TDC LUT 2 -> 31	RO
36	0x2400	TDC 32 LUT	RO
37	0x2500	TDC 33 LUT	RO
38	0x2600	Serial Flash Control	RW + RO
39	0x2700	Remote Update Control	RW + RO
40	0x2800	TDC Timestamp Correction	RW
41	0x2900	Data Path Control	RW

FPGA General Control

This slave has mainly a debug, verification and identification purpose.

The 16 first registers of this slaves have a RW access but their value is ignored internally. This feature allows to safely testing the slow control communication with the FPGAs.

This is also possible to read the FPGA ID for each of the three FPGAs of the board (0,1,2 depending on their location), the firmware revision ID and the FPGA chip ID.

Register Address		Register Name	Bit range	Description	Access	Default Value
0	0x00	RW TEST REG	[15:0]	IGNORED	RW	0x0000
…	…		[15:0]			
15	0x0F		[15:0]			
16	0x10	FPGA INFO	[1:0]	FPGA_ID	RO	N/A
			[15:2]	NOT USED (zeroed)		
17	0x11	FIRMWARE INFO	[15:0]	FW_MASTER_REV_ID	RO	N/A
18	0x12		[15:0]	FW_MINOR_REV_ID	RO	N/A
19	0x13	FPGA CHIP ID	[15:0]	Chip ID [63:48]	RO	N/A
20	0x14		[15:0]	Chip ID [47:32]	RO	N/A
21	0x15		[15:0]	Chip ID [31:16]	RO	N/A
22	0x16		[15:0]	Chip ID [15:0]	RO	N/A

PETIROC Control

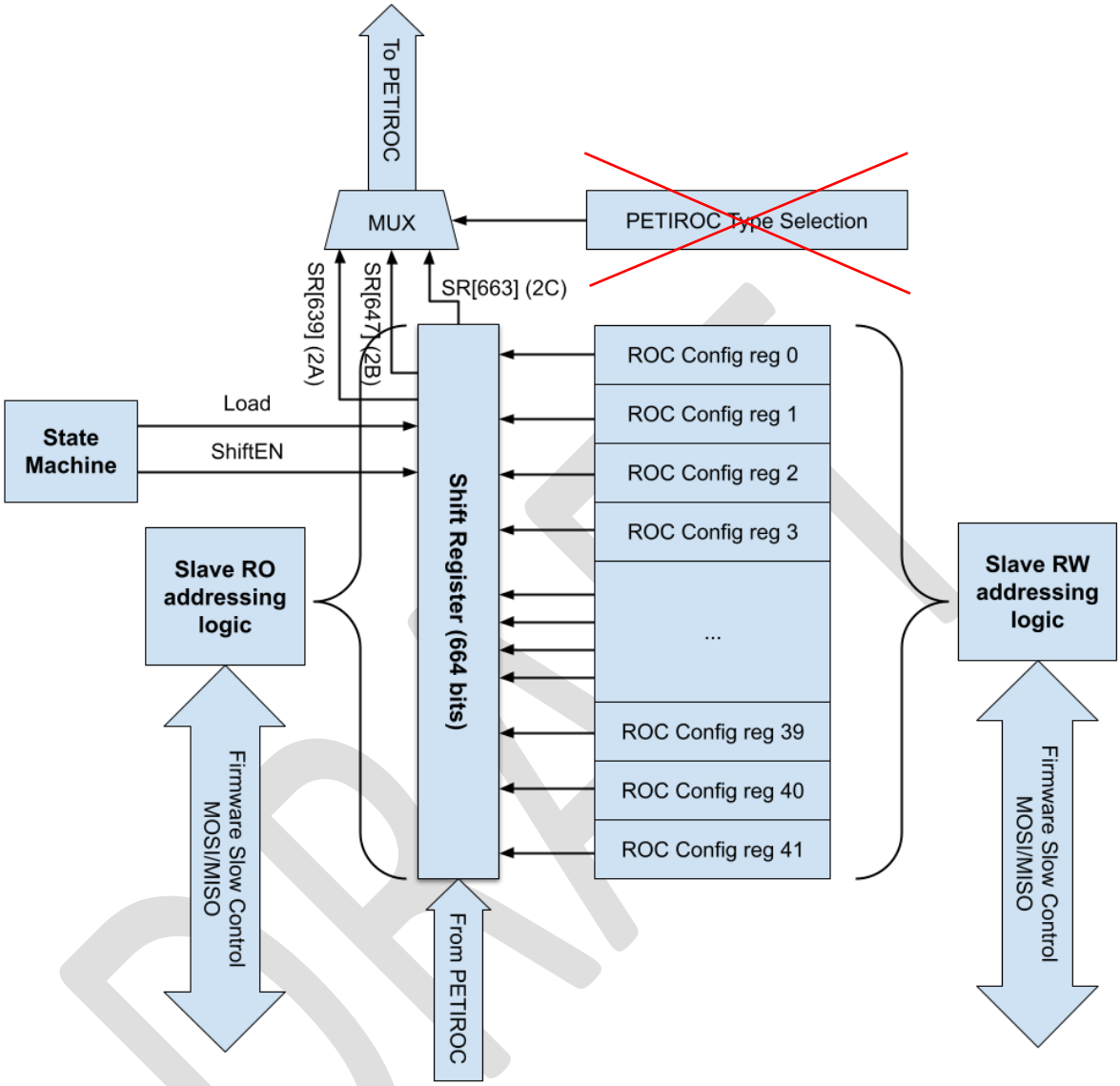
In addition to the control and status registers, this slave hosts a state machine dedicated to command the PETIROC slow control interface. This state machine is able to perform the reset sequence or the loading sequence (serialization) of the parameters in the PETIROC slow control shift register.

Register Address		Register Name	Bit range	Description	Access	Default Value
0	0x00	PETIROC LOAD CTRL	[0]	'1' -> PETIROC configuration sequence request	RW	0x0000
			[1]	'1' -> Enable the PETIROC periodic reconfiguration		
			[15:2]	IGNORED		
1	0x01	PETIROC RESET CTRL	[0]	'1' -> PETIROC reset sequence request	RW	0x0000
			[15:1]	IGNORED		
2	0x02	PETIROC slow control pins	[0]	'1' -> Force PETIROC digital stage OFF	RW	0x0000
			[1]	'1' -> Force PETIROC analog stage OFF		
			[2]	'1' -> Force PETIROC ADC stage OFF		
			[3]	'1' -> Force PETIROC DAC stage OFF		
			[4]	Active low PETIROC digital part reset		
			[15:5]	IGNORED		
3	0x03	PETIROC injection pins	[0]	'1' -> Emulate an input signal for every channels of the PETIROC	RW	0x0000
			[1]	Connected to the hold_ext pin of the PETIROC		
			[15:2]	IGNORED		
...	..	NOT USED				
9	0x09	Bitflip counter reset	[0]	'1' -> Reset the bitflip counter	RW	0x0000
			[15:1]	IGNORED		
10	0x0A	PETIROC auto-reconfiguration period (LSb = 8.33 ns)	[15:0]	Reconfiguration period[15:0]	RW	0x0000
11	0x0B		[15:0]	Reconfiguration period[31:16]	RW	0x0000
12	0x0C		[15:0]	Reconfiguration period[47:32]	RW	0x0000
...	..	NOT USED				
22	0x16	ConfigToROC Word0	[15:0]	To PETIROC Internal Register [663:648]	RW	0x0000
23	0x17	ConfigToROC Word1	[15:0]	To PETIROC Internal Register [647:632]	RW	0x0000
24	0x18	ConfigToROC Word2	[15:0]	To PETIROC Internal Register [631:616]	RW	0x0000
...	...	ConfigToROC Words...	[15:0]	To PETIROC Internal Register [615:24]	RW	0x0000
62	0x3E	ConfigToROC Word40	[15:0]	To PETIROC Internal Register [23:8]	RW	0x0000
63	0x3F	ConfigToROC Word41	[15:8]	To PETIROC Internal Register [7:0]	RW	0x0000
			[7:0]	IGNORED		
64	0x40	Configuration bitflip counter	[15:0]	Bitflip counter[15:0]	RO	N/A
65	0x41		[15:0]	Bitflip counter[31:16]	RO	N/A
...	..	NOT USED				
99	0x63	ShiftRegWord0	[15:0]	From PETIROC Internal Register [663:648]	RO	N/A
100	0x64	ShiftRegWord1	[15:0]	From PETIROC Internal Register [647:632]	RO	N/A
101	0x65	ShiftRegWord2	[15:0]	From PETIROC Internal Register [631:616]	RO	N/A
...	...	ShiftRegWords...	[15:0]	From PETIROC Internal Register [615:24]	RO	N/A
139	0x8B	ShiftRegWord40	[15:0]	From PETIROC Internal Register [23:8]	RO	N/A
140	0x8C	ShiftRegWord41	[15:8]	From PETIROC Internal Register [7:0]	RO	N/A
			[7:0]	NOT USED (zeroed)		

Warning: Since Firmware v3, the LSb of the PETIROC auto-reconfiguration period is now 8.33ns (clk120MHz period) instead of 10ns.

PETIROC Slow control loading block diagram

WARNING: PETIROC Type selection feature is removed since FW v4.0 (fixed to PETIROC 2C)



TDC Control

This slave hosts registers dedicated to control the TDC related features.

Register Address		Register Name	Register range	Description	Access	Default Value
0	0x00	TDC Enable	[0]	'1'-> Enable the TDC module	RW	0x0000
			[15:1]	IGNORED		
1	0x01	CMD Valid	[0]	'1' -> Specify the state of the CMD registers is ready to be registered	RW	0x0000
			[15:1]	IGNORED		
2	0x02	CMD Cali Req (Chan 0->15)	[15:0]	'1' -> TDC calibration request for each channel	RW	0x0000
3	0x03	CMD Cali Req (Chan 16->31)	[15:0]			0x0000
4	0x04	CMD Cali Req (Chan 32->33)	[1:0]			0x0000
			[15:2]	IGNORED		
5	0x05	CMD Meas Enable (Chan 0->15)	[15:0]	'1' -> TDC Measure enable for each channel	RW	0x0000
6	0x06	CMD Meas Enable (Chan 16->31)	[15:0]			0x0000
7	0x07	CMD Meas Enable (Chan 32->33)	[1:0]			0x0000
			[15:2]	IGNORED		
8	0x08	Injection Mode	[3:0]	TDC Injection mode selection	RW	0x0000
			[15:4]	IGNORED		
9	0x09	Injection BC0 timing offset	[15:0]	Time offset between the reception of BC0 and trigger injection	RW	0x0000
10	0x0A	DataCounterTimeWindow	[15:0]	DataCounterTimeWindow[15:0]	RW	0x0000
11	0x0B	(LSb = 8.33 ns)	[15:0]	DataCounterTimeWindow[31:16]	RW	0x0000
12	0x0C	DataCounter CTRL	[0]	'1' -> Start data counting	RW	0x0000
			[15:1]	IGNORED		
13	0x0D	BC0 feature control	[0]	'0' -> Raw timestamps | '1' -> Enable BC0 Offset Correction	RW	0x0001
			[1]	'0' -> BC0 channels are forwarded | '1' -> BC0 timestamps dropped		
			[15:2]	IGNORED		
...	..	NOT USED				
16	0x10	DNL Build Done (Chan 0->15)	[15:0]	TDC DNL build done for each channel	RO	N/A
17	0x11	DNL Build Done (Chan 16->31)	[15:0]			
18	0x12	DNL Build Done (Chan 32->33)	[1:0]			
			[15:2]	NOT USED (zeroed)		
19	0x13	LUT Build Done (Chan 0->15)	[15:0]	TDC LUT build done for each channel	RO	N/A
20	0x14	LUT Build Done (Chan 16->31)	[15:0]			
21	0x15	LUT Build Done (Chan 32->33)	[1:0]			
			[15:2]	NOT USED (zeroed)		
22	0x16	Data Counter valid	[0]	'1' -> data counting sequence is over, data counters can be read	RO	N/A
			[15:1]	NOT USED (zeroed)		
23	0x17	DataCounter Chan0	[15:0]	DataCounter Chan0 [15:0]	RO	N/A
24	0x18		[15:0]	DataCounter Chan0 [31:16]	RO	N/A
25	0x19	DataCounter Chan1	[15:0]	DataCounter Chan1 [15:0]	RO	N/A
26	0x1A		[15:0]	DataCounter Chan1 [31:16]	RO	N/A
...	...	DataCounter Chan 2 -> 32	[15:0]		RO	N/A
89	0x59	DataCounter Chan33	[15:0]	DataCounter Chan33 [15:0]	RO	N/A
90	0x5A		[15:0]	DataCounter Chan33 [31:16]	RO	N/A

Injection Mode mapping:

- “0000” => Standard mode (PETIROC outputs for the first 32 TDC channels).
- “0001” => An independent 1kHz clock is applied as input for every TDC channels.
- “0010” => Upon reception of a BC0, a delayed pulse is broadcasted to the first 32 TDC channels (delay is configurable with the Injection BC0 timing offset).
- “0100” => Upon reception of a BC0, activate the trig_ext pin for the 2 linked PETIROCs.
- “1000” => Upon reception of a Resync, activate the trig_ext pin for the 2 linked PETIROCs

Warning: Since Firmware v3.0, the LSb of the Data Counter Time Window value is now 8.33ns (clk120MHz period) instead of 10ns.

TDC LUT Readout

It is possible to read the content of the TDC LUTs which are generated during the calibration procedure. All of these LUTs are independent slaves connected to the internal slow control bus. Access to these LUTs is possible in read-only.

Serial Flash Control

This slave ensures the communication with the Flash EEPROM connected to the FPGA. The memory is divided into 256 sectors of 65536 bytes. The first 128 sectors are allocated to the Golden Firmware, the second part of the memory is allocated for the Application Firmware. The flash control operations such as sector protection and sector erasing are done using the CSR interface while the memory content read and write operation are managed by the Memory interface.

Register Address		Register Name	Register range	Description	Access	Default Value
0	0x00	CSR_WriteData	[15:0]	csr_writedata[31:16]	RW	0x0000
1	0x01		[15:0]	csr_writedata[15:0]	RW	0x0000
2	0x02	CSR_Address	[5:0]	csr_address[5:0]	RW	0x0000
			[15:6]	IGNORED		
3	0x03	CSR_Write	[0]	csr_write (this bit is automatically reset internally)	RW	0x0000
			[15:1]	IGNORED		
4	0x04	CSR_Read	[0]	csr_read (this bit is automatically reset internally)	RW	0x0000
			[15:1]	IGNORED		
5	0x05	mem_burstcount	[6:0]	mem_burstcount[6:0]	RW	0x0000
			[15:7]	IGNORED		
6	0x06	mem_byteenable	[3:0]	mem_byteenable[3:0]	RW	0x0000
			[15:4]	IGNORED		
7	0x07	mem_WriteData word0	[15:0]	mem_writedata[31:16]	RW	0x0000
8	0x08		[15:0]	mem_writedata[15:0]	RW	0x0000
9	0x09	mem_WriteData word1	[15:0]	mem_writedata[31:16]	RW	0x0000
10	0x0A		[15:0]	mem_writedata[15:0]	RW	0x0000
...	...	mem_WriteData words 2 -> 29	[15:0]	mem_writedata[31:16]	RW	0x0000
			[15:0]	mem_writedata[15:0]	RW	0x0000
67	0x43	mem_WriteData word30	[15:0]	mem_writedata[31:16]	RW	0x0000
68	0x44		[15:0]	mem_writedata[15:0]	RW	0x0000
69	0x45	mem_WriteData word31	[15:0]	mem_writedata[31:16]	RW	0x0000
70	0x46		[15:0]	mem_writedata[15:0]	RW	0x0000
71	0x47	mem_address	[5:0]	mem_address[21:16]	RW	0x0000
			[15:6]	IGNORED		
72	0x48		[15:0]	mem_address[15:0]	RW	0x0000
73	0x49	mem_Write	[0]	mem_write (this bit is automatically reset internally)	RW	0x0000
			[15:1]	IGNORED		
74	0x4A	mem_Read	[0]	mem_read (this bit is automatically reset internally)	RW	0x0000
			[15:1]	IGNORED		
...	..	NOT USED				
128	0x80	CSR_WaitRequest	[0]	csr_waitrequest	RO	N/A
			[15:1]	NOT USED (zeroed)		
129	0x81	CSR_ReadData	[15:0]	csr_readdata[31:16]	RO	N/A
130	0x82		[15:0]	csr_readdata[15:0]	RO	N/A
131	0x83	mem_WaitRequest	[0]	mem_waitrequest	RO	N/A
			[15:1]	NOT USED (zeroed)		
132	0x84	mem_ReadData word0	[15:0]	mem_readdata[31:16]	RO	N/A
133	0x85		[15:0]	mem_readdata[15:0]	RO	N/A
134	0x86	mem_ReadData word1	[15:0]	mem_readdata[31:16]	RO	N/A
135	0x87		[15:0]	mem_readdata[15:0]	RO	N/A
...	...	mem_ReadData words 2 -> 29	[15:0]	mem_readdata[31:16]	RO	N/A
			[15:0]	mem_readdata[15:0]	RO	N/A
192	0xC0	mem_ReadData word30	[15:0]	mem_readdata[31:16]	RO	N/A
193	0xC1		[15:0]	mem_readdata[15:0]	RO	N/A
194	0xC2	mem_ReadData word31	[15:0]	mem_readdata[31:16]	RO	N/A
195	0xC3		[15:0]	mem_readdata[15:0]	RO	N/A

CSR interface

This interface allows to perform control functions over the EEPROM linked to the FPGA such as sector protection and sector erasing. As this device is a NOR Flash memory, a sector needs to be erased before to be rewritten (erasing a sector sets all its bits at '1'). These actions are mandatory to edit (update) the Application Firmware located in the second half of the flash memory.

Write a CSR register

To write a CSR register, as the 4 Slow Control registers you need to write are contiguous, the easiest way to proceed is to send the following GBT frame burst write request:

Burst WRITE request of 4 data words (BurstAdditionnalWords=3):

REQUEST ADDRESS => 0x2600

REQUEST PAYLOAD => [csr_writedata[31:16], csr_writedata[15:0], csr_address, “1”]

Alternatively, you can send the parameters in 4 single word write request frames (addressed to 0x2600, 0x2601, 0x2602 and 0x2603).

In any case, take note that the writing of “1” in the register 0x2603 has to occur last (as this bit launches the register writing process).

Read a CSR register

To read the value of a CSR register, the easiest way to proceed is to send the following GTB frame burst write request:

Burst WRITE request of 3 data words (BurstAdditionnalWords=2):

REQUEST ADDRESS => 0x2602

REQUEST PAYLOAD => [csr_address, “0”, ”1”]

Alternatively, you can send 2 single word write request frames (addressed to 0x2602 and 0x2604). In any case, take note the writing of “1” in the register 0x2604 has to occur last (as this bit launches the register reading process).

The return value is stored in the FPGA registers 0x2681 and 0x2682, you can easily read it using the following burst read request:

Burst READ request of 2 words (BurstAdditionnalWords=1):

REQUEST ADDRESS => 0x2681

REPLY PAYLOAD => [csr_readdata[31:16], csr_readdata[15:0]]

Set protection to sectors [0;127] and unprotect sectors [128;255]

This operation allows edition of the application firmware part (second half of the memory) while keeping safe the golden firmware part (first half of the memory) of the flash.

This procedure uses the function defined in the “Write a CSR register” section.

---

flash_csr_write(csr_writedata = 0x6, csr_address = 0x7)

flash_csr_write(csr_writedata = 0x01, csr_address = 0x8)

wait 50ms

flash_csr_write(csr_writedata = 0x1001, csr_address = 0x7)

flash_csr_write(csr_writedata = 0x38, csr_address = 0xA)

flash_csr_write(csr_writedata = 0x01, csr_address = 0x8)

wait 100ms

Erase a sector of the flash memory

This operation turns every memory bits of a given sector at “1”, making possible to write new data (a memory write operation can only turns “1” into “0”). If the sector is protected, this should have no effect. This procedure uses the function defined in the “Write a CSR register” section.

---

flash_csr_write(csr_writedata = 0x6, csr_address = 0x7)

flash_csr_write(csr_writedata = 0x01, csr_address = 0x8)

wait 50ms

flash_csr_write(csr_writedata = (nSector<<16), csr_address = 0x9)

flash_csr_write(csr_writedata = 0x3D8, csr_address = 0x7)

flash_csr_write(csr_writedata = 0x01, csr_address = 0x8)

wait 500ms

“nSector<<16” means the sector ID shifted left of 16bits, this represents the byte base address of a given sector (as a sector is composed of 65536=2^16 bytes).
This procedure has to be performed 128 times, 1 time for each sector of the Application Firmware part.

Summary of the CSR interface utilization

In order to update the Application Firmware stored in the flash memory, it is mandatory to use the CSR interface to perform these preliminary steps:
Protect sectors [0:127] -> This ensures the Golden Firmware part remains untouched.
Unprotect sectors [128:255] -> This allows the edition of the flash content where the Application Firmware is stored.
Erase sectors [128:255] -> This reset every bit of the Application Firmware part to “1”, allowing to write a new content using the memory interface.

Memory interface

This interface allows to read the content and to write a new content in the flash memory. The information needed to write a new Application Firmware in the flash memory is located in the Quartus output files .rpd (raw binary flash content) and .map (gives the firmware size).

Important note

The memory interface addressing system is based on 32bits words. This explains why the addresses used in this part are shifted right of 2 bits compared to the addresses used in the CSR interface part.

Example:
Byte base address of the Application Firmware => 0x800000 (used with CSR interface)
Word base address of the Application Firmware => 0x200000 (used with memory interface)

Write content of the flash

Thanks to the management of the burst mechanism of the interface, it is possible to write up to 32 memory words using only one GBT frame transaction:

Burst WRITE request of 69 data words (BurstAdditionnalWords=68):

REQUEST ADDRESS => 0x2605

REQUEST PAYLOAD => [32, 15, word0MSB, word0LSB, …, word31MSB, word31LSB, addrMSB, addrLSB, “1”]

Here is the detail of the different fields of the GBT frame transaction payload:

- 32 refers to a burst length of 32 memory words at the interface level.
- 15 = 0xF refers to a byte mask at the interface value, it should be always set to this value.
- wordnMSB, wordnLSB refer to a memory word (32 bits) which is split in two halves because the FPGA SlowControl communication is based on 16 bits words.
- addrMSB, addrLSB refer to the memory word base address of the burst write request.
- “1” is to request the launch of the memory words writing into the flash.

Read content of the flash

Thanks to the management of the burst mechanism of the interface, it is possible to read up to 32 memory words using only one GBT frame transaction:

Burst WRITE request of 70 data words (BurstAdditionnalWords=69):

REQUEST ADDRESS => 0x2605

REQUEST PAYLOAD => [32, 15, (0, 0…,0, 0 -> 64 words at 0), addrMSB, addrLSB, “0”, “1”]

The return value (the content of the 32 memory words) is then available in the 64 registers from 0x2684 to 0x26C3. They can be read easily using the following GBT frame read burst request:

Burst READ request of 64 words (BurstAdditionnalWords=63):

REQUEST ADDRESS => 0x2684

REPLY PAYLOAD => [word0MSB, word0LSB, … , word31MSB, word31LSB]

Summary of the memory interface utilization

This interface can be used to write 32 memory words (each composed of 4 bytes) in a single transaction (this means that 512 consecutive operations are needed to fully rewrite a flash sector).

Remote update controller

This slave allows to jump from the golden firmware (which is the default boot firmware) to the application firmware (the firmware which can be remotely updated without risk).

Register Address		Register Name	Register range	Description	Access	Default Value
0	0x00	CSR_WriteData	[15:0]	csr_writedata[31:16]	RW	0x0000
1	0x01		[15:0]	csr_writedata[15:0]	RW	0x0000
2	0x02	CSR_Address	[2:0]	csr_address[2:0]	RW	0x0000
			[15:3]	IGNORED		
3	0x03	CSR_Write	[0]	csr_write (this bit is automatically reset internally)	RW	0x0000
			[15:1]	IGNORED		
4	0x04	CSR_Read	[0]	csr_read (this bit is automatically reset internally)	RW	0x0000
			[15:1]	IGNORED		
...	..	NOT USED				
16	0x10	CSR_WaitRequest	[0]	csr_waitrequest	RO	N/A
			[15:1]	NOT USED (zeroed)		
17	0x11	CSR_ReadData	[15:0]	csr_readdata[31:16]	RO	N/A
18	0x12		[15:0]	csr_readdata[15:0]	RO	N/A

Write a CSR register

To write a CSR register, as the 4 SlowControl registers you need to write are contiguous, the easiest way to proceed is to send the following GBT frame burst write request:

Burst WRITE request of 4 data words (BurstAdditionnalWords=3):

REQUEST ADDRESS => 0x2700

REQUEST PAYLOAD => [csr_writedata[31:16], csr_writedata[15:0], csr_address, “1”]

Alternatively, you can send the parameters in 4 single word write request frames (addressed to 0x2700, 0x2701, 0x2702 and 0x2703).

In any case, take note that the writing of “1” in the register 0x2703 has to occur last (as this bit launches the register writing process).

Read a CSR register

To read the value of a CSR register, the easiest way to proceed is to send the following GTB frame burst write request:

Burst WRITE request of 3 data words (BurstAdditionnalWords=2):

REQUEST ADDRESS => 0x2702

REQUEST PAYLOAD => [csr_address, “0”, ”1”]

Alternatively, you can send 2 single word write request frames (addressed to 0x2702 and 0x2704).
In any case, take note the writing of “1” in the register 0x2704 has to occur last (as this bit launches the register reading process).

The return value is stored in the FPGA registers 0x2711 and 0x2712, you can easily read it using the following burst read request:

Burst READ request of 2 words (BurstAdditionnalWords=1):

REQUEST ADDRESS => 0x2711

REPLY PAYLOAD => [csr_readdata[31:16], csr_readdata[15:0]]

Jump from Golden Firmware to Application Firmware

To perform the jump to the Application Firmware, one needs to send the instructions to the Remote Update Controller through the CSR interface (especially the byte base address of the Application Firmware which is 0x800000). After the firmware jump, a SCA GPIO driven SOFT_RESET cycle is needed to resynchronize the communication links (with the GBTx and between FPGAs). Unlike a N_CONFIG cycle, a SOFT_RESET doesn't change the firmware.

This procedure uses the functions defined in the "Write a CSR register" and "Read a CSR register" sections.

```
rem_csr_write(csr_writedata =  0x800000, csr_address = 0x3)   #Set AppFW base address
rem_csr_write(csr_writedata =  0x1, csr_address = 0x4)        #Set FW ctrl bit to "1"
rem_csr_write(csr_writedata =  0x1, csr_address = 0x6)        #FW Jump request
wait 500ms
Perform a SOFT_RESET cycle (using the SCA GPIO control) #To get back communication with FPGA
SUCCESS = rem_csr_read(csr_address = 0x4)                     #Check the FW ctrl bit value
```

If the return value SUCCESS is "1", then the jump is a success and the FPGA is now working using the Application Firmware. This firmware will remain active until a FPGA power cycle or a N_CONFIG cycle (which always asks the FPGA to load the Golden Firmware).

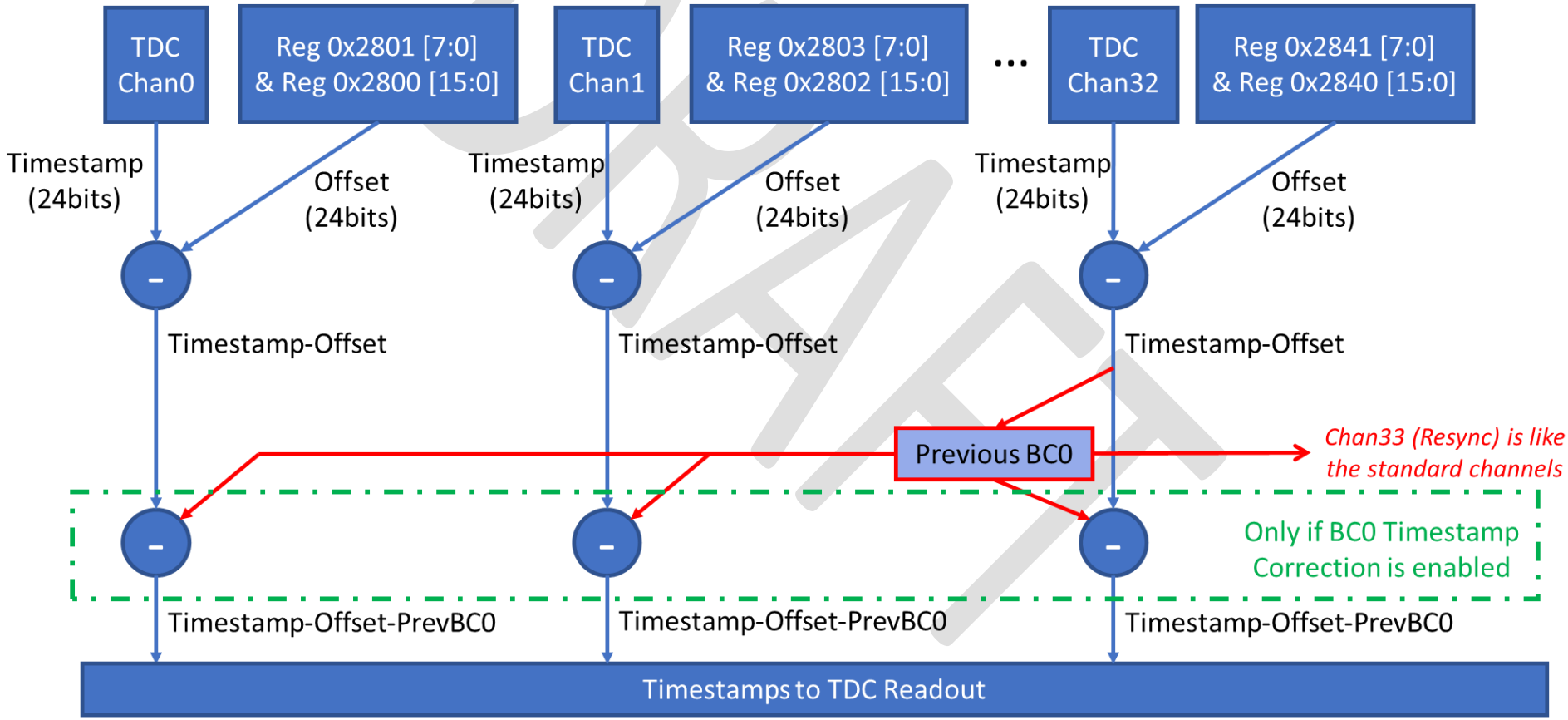
If the return value SUCCESS is "0", then the jump is a fail and the FPGA is back to the Golden Firmware. This is probably caused by a corruption of the data located in the Application Firmware part.

TDC Timestamp correction module

This module provides a configurable timestamp offset correction for each of the 34 TDC channels.

Register	Address	Register Name	Register range	Description	Access	Default Value
0	0x00	TDC Chan0 Timestamp correction	[15:0]	Timestamp correction[15:0]	RW	0x0000
1	0x01		[7:0]	Timestamp correction[23:16]	RW	0x0000
			[15:8]	IGNORED		
2	0x02	TDC Chan1 Timestamp correction	[15:0]	Timestamp correction[15:0]	RW	0x0000
3	0x03		[7:0]	Timestamp correction[23:16]	RW	0x0000
			[15:8]	IGNORED		
4	0x04	TDC Chan2 Timestamp correction	[15:0]	Timestamp correction[15:0]	RW	0x0000
5	0x05		[7:0]	Timestamp correction[23:16]	RW	0x0000
			[15:8]	IGNORED		
6	0x06	TDC Chan3 Timestamp correction	[15:0]	Timestamp correction[15:0]	RW	0x0000
7	0x07		[7:0]	Timestamp correction[23:16]	RW	0x0000
			[15:8]	IGNORED		
…	…	TDC Chan 4 -> 29 Timestamp correction	[15:0]	Timestamp correction[15:0]	RW	0x0000
			[7:0]	Timestamp correction[23:16]	RW	0x0000
			[15:8]	IGNORED		
60	0x3C	TDC Chan30 Timestamp correction	[15:0]	Timestamp correction[15:0]	RW	0x0000
61	0x3D		[7:0]	Timestamp correction[23:16]	RW	0x0000
			[15:8]	IGNORED		
62	0x3E	TDC Chan31 Timestamp correction	[15:0]	Timestamp correction[15:0]	RW	0x0000
63	0x3F		[7:0]	Timestamp correction[23:16]	RW	0x0000
			[15:8]	IGNORED		
64	0x40	TDC Chan32 (BC0) Timestamp correction	[15:0]	Timestamp correction[15:0]	RW	0x0000
65	0x41		[7:0]	Timestamp correction[23:16]	RW	0x0000
			[15:8]	IGNORED		
66	0x42	TDC Chan33 (Resync) Timestamp correction	[15:0]	Timestamp correction[15:0]	RW	0x0000
67	0x43		[7:0]	Timestamp correction[23:16]	RW	0x0000
			[15:8]	IGNORED		

TDC timestamp correction module
(Channel offset SlowControl slave base address -> 0x2800)



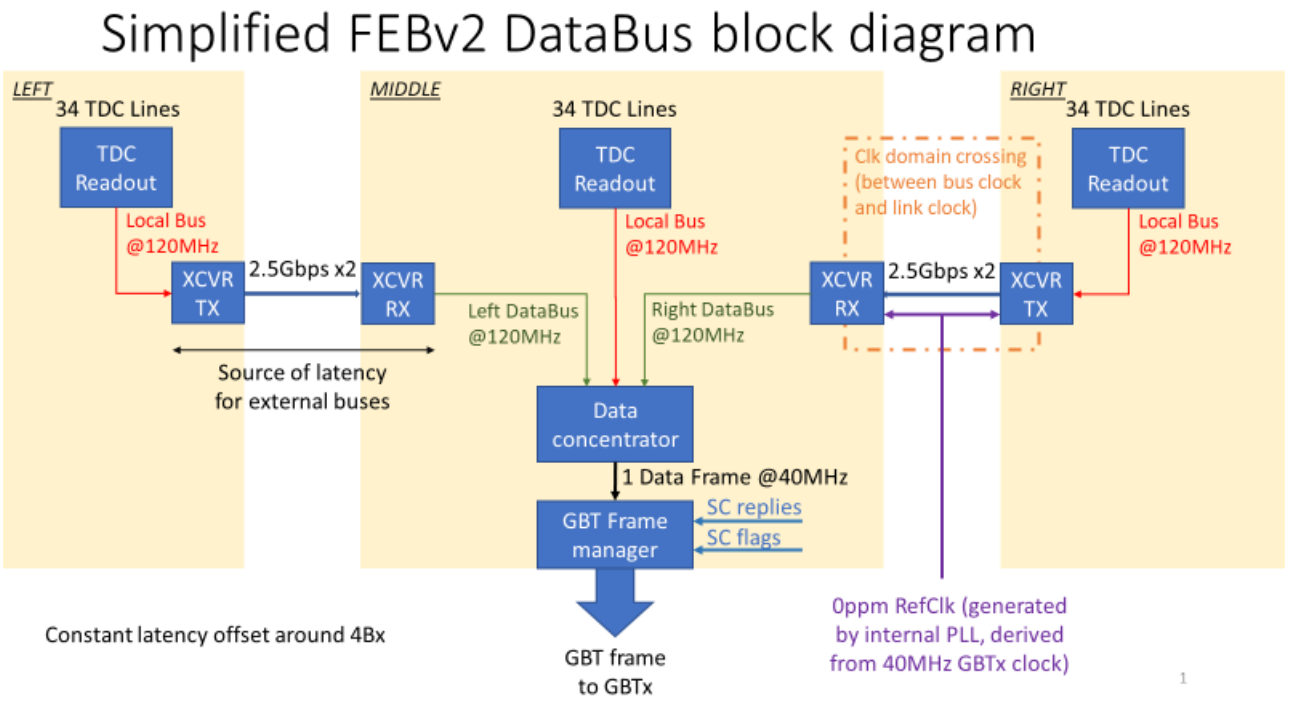
Data Path Control

This slave allows to configure the different features related to data path flow and latency regulation.

Register Address		Register Name	Register range	Description	Access	Default Value	
0	0x00	Middle FPGA Bus Extra Delay	[5:0]	Add an extra-latency on data from Middle FPGA (in number of CLK120MHz cycles -> LSb = 8,33ns)	RW	0x000D	Data Concentrator configuration Only used in the Middle FPGA
			[15:6]	IGNORED			
1	0x01	Output Frame Queue Maximum Size	[5:0]	Maximum queue length for output data frames (in number of frames = CLK40MHz cycles)	RW	0x003F	
			[15:6]	IGNORED			
2	0x02	Strip Clustering Enable	[0]	'1' -> Enable merging of the 2 channels from the same strip	RW	0x0000	
			[15:1]	IGNORED			
3	0x03	Remove Single Channel Data	[0]	'1'-> Drop single channel data from FPGA 0	RW	0x0000	
			[1]	'1'-> Drop single channel data from FPGA 1			
			[2]	'1'-> Drop single channel data from FPGA 2			
			[15:3]	IGNORED			
4	0x04	Readout Maximum Time Disparity	[6:0]	Maximum time drift between data creation and readout (in number of CLK120MHz cycles -> LSb = 8,33ns)	RW	0x007F	TDC Readout configuration Have to be configured for the 3 FPGAs
			[15:7]	IGNORED			
5	0x05	Channel Dead Time	[5:0]	Minimum time between 2 data produced from the same TDC channel (in number of CLK120MHz cycles -> LSb = 8,33ns)	RW	0x0000	
			[15:6]	IGNORED			
6	0x06	Pair Filtering Enable	[0]	'1' -> Pair Filtering Enable Strip 0	RW	0x0000	
			[1]	'1' -> Pair Filtering Enable Strip 1			
			[N]	'1' -> Pair Filtering Enable Strip N			
			[14]	'1' -> Pair Filtering Enable Strip 14			
			[15]	'1' -> Pair Filtering Enable Strip 15			
7	0x07	NOT USED					
8	0x08	Pair TS Diff Min Strip 0	[15:0]	Minimum time difference between the 2 ends of the Strip 0	RW	0x0000	
9	0x09	Pair TS Diff Min Strip 1	[15:0]	Minimum time difference between the 2 ends of the Strip 1	RW	0x0000	
...	..	Pair TS Diff Min Strip N	[15:0]	Minimum time difference between the 2 ends of the Strip N	RW	0x0000	
22	0x16	Pair TS Diff Min Strip 14	[15:0]	Minimum time difference between the 2 ends of the Strip 14	RW	0x0000	
23	0x17	Pair TS Diff Min Strip 15	[15:0]	Minimum time difference between the 2 ends of the Strip 15	RW	0x0000	
24	0x18	Pair TS Diff Max Strip 0	[15:0]	Maximum time difference between the 2 ends of the Strip 0	RW	0xFFFF	
25	0x19	Pair TS Diff Max Strip 1	[15:0]	Maximum time difference between the 2 ends of the Strip 1	RW	0xFFFF	
...	..	Pair TS Diff Max Strip N	[15:0]	Maximum time difference between the 2 ends of the Strip N	RW	0xFFFF	
38	0x26	Pair TS Diff Max Strip 14	[15:0]	Maximum time difference between the 2 ends of the Strip 14	RW	0xFFFF	
39	0x27	Pair TS Diff Max Strip 15	[15:0]	Maximum time difference between the 2 ends of the Strip 15	RW	0xFFFF	
40	0x28	Retrig mitigation counter threshold	[3:0]	Counter threshold (when the amount of data at the input of 1 channel is above this threshold, the retriggering mitigation procedure is launched)	RW	0x0000	
			[15:4]	IGNORED			
41	0x29	Retrig mitigation counter decrement time	[7:0]	Period between 2 decrement of the retriggering data counter (in number of CLK120MHz cycles -> LSb = 8,33ns)	RW	0x0000	
			[15:8]	IGNORED			
42	0x2A	Retrig mitigation MUTE_ROC duration	[7:0]	Mute the 2 PETIROCs for the specified time after a retrig detection (in number of CLK120MHz cycles -> LSb = 8,33ns)	RW	0x0000	
			[15:8]	IGNORED			

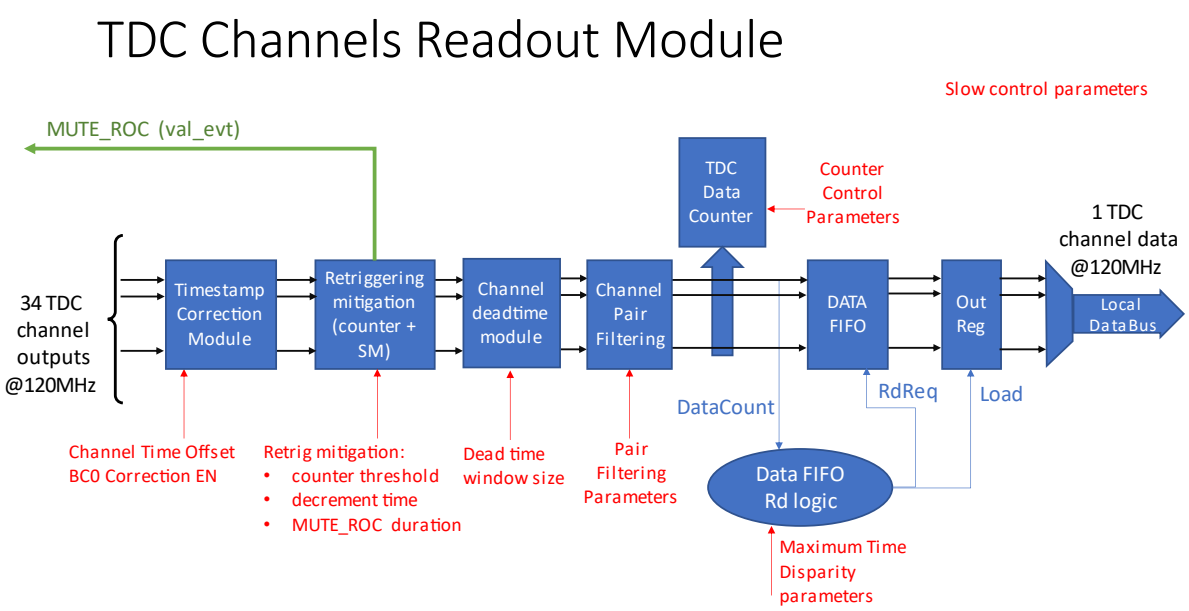
Registers 0 to 3 are dedicated to the data concentrator / GBT frame creator located in the Middle FPGA, these registers only have an effect in FPGA 1.

The other registers are dedicated to the configuration of the TDC Readout module located in each FPGA.



TDC Channels Readout Module

This module is intended to read incoming data from the 34 TDC channels, to filter unexpected data and to organize data to feed the local data bus in each of the 3 FPGAs.



Timestamp correction module

This module is intended to convert the raw timestamp from TDC into the expected timestamp. This is done by subtracting the last BC0 raw timestamp (the channel time refers to the time between the BC0 and the detected hit) and subtracting a configurable constant offset (to get rid of the time skew between channels related to the RPC strip geometry).

Retriggering mitigation module

When the threshold of the PETIROC is too low (too close from noise), a retriggering effect could appear. This effect causes an oscillation of one or many PETIROC channels and therefore produces a lot of consecutive false triggers after the first real one. As each consecutive trigger also creates a new timestamp in the TDC module, this causes a flood of unexpected data in the readout module.

The retriggering mitigation module is here to detect when a PETIROC channel is oscillating (a high number of timestamps are generated within a small time) and to mute the PETIROC output for a given configurable time, allowing it to recover from its oscillating state.

Channel dead time module

The channel dead time module is a filter which ensures a minimum delay between 2 data generated by the same TDC channel. Increasing the deadtime of the channels allows to hide the retriggering effect by dropping the timestamps that are due to consecutive triggers.

Channel pair filtering module

The channel pair filtering module allows to validate the data matching between the 2 channels of a strip. The timestamp difference is processed and data are dropped if the value is not within the acceptable range (configurable minimum and maximum). If the value is correct, the data from the 2 channels are then recorded in the data buffer at the same time. Take note that when the pair filtering is enabled, the first incoming data of the channel pair (from the direct strip channel) is delayed until the arrival of the second data of the pair (from the return strip channel). This effect can lead to a light increase of data latency.

Data from Resync and BC0 channels are not affected by this module and are just forwarded in the pipeline as soon as they come.

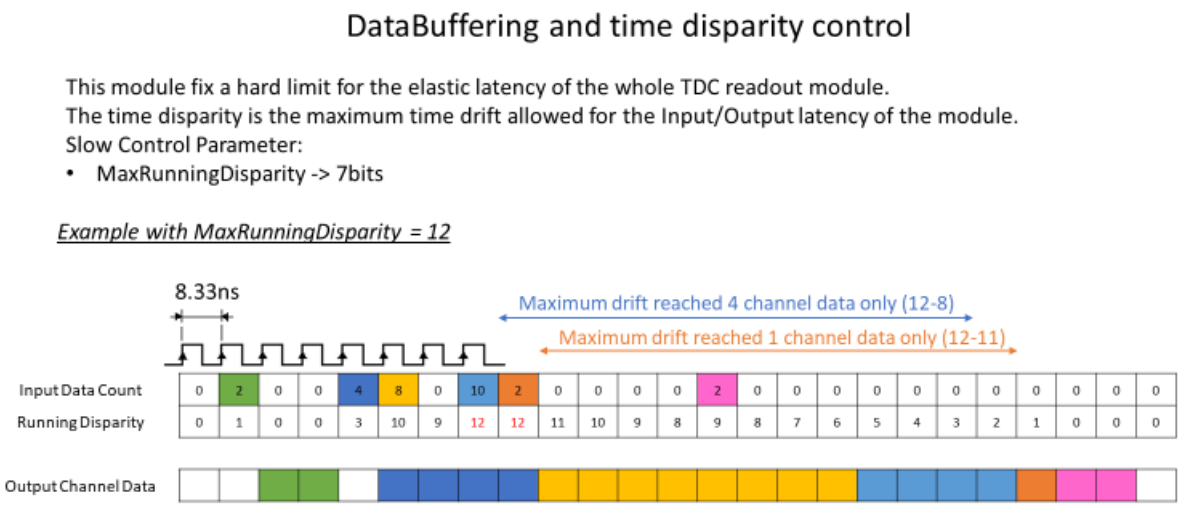
Data buffering and multiplexing

The data buffer is made of a FIFO with a mechanism of maximum time disparity control. This allows to set a maximum time budget for the data multiplexing in the 3 FPGAs.

The multiplexing is made using a constant priority encoder which respects the following rule (from most to least priority):

TDC Channel ID	Source	
33	Resync	**Sync signals**
32	BC0	
15	Direct Strip num = FPGA_ID*16	**Even Strips**
16	Return Strip num = FPGA_ID*16	
13	Direct Strip num = FPGA_ID*16 + 2	
18	Return Strip num = FPGA_ID*16 + 2	
11	Direct Strip num = FPGA_ID*16 + 4	
20	Return Strip num = FPGA_ID*16 + 4	
9	Direct Strip num = FPGA_ID*16 + 6	
22	Return Strip num = FPGA_ID*16 + 6	
7	Direct Strip num = FPGA_ID*16 + 8	
24	Return Strip num = FPGA_ID*16 + 8	
5	Direct Strip num = FPGA_ID*16 + 10	
26	Return Strip num = FPGA_ID*16 + 10	
3	Direct Strip num = FPGA_ID*16 + 12	
28	Return Strip num = FPGA_ID*16 + 12	
1	Direct Strip num = FPGA_ID*16 + 14	
30	Return Strip num = FPGA_ID*16 + 14	
14	Direct Strip num = FPGA_ID*16 + 1	**Odd Strips**
17	Return Strip num = FPGA_ID*16 + 1	
12	Direct Strip num = FPGA_ID*16 + 3	
19	Return Strip num = FPGA_ID*16 + 3	
10	Direct Strip num = FPGA_ID*16 + 5	
21	Return Strip num = FPGA_ID*16 + 5	
8	Direct Strip num = FPGA_ID*16 + 7	
23	Return Strip num = FPGA_ID*16 + 7	
6	Direct Strip num = FPGA_ID*16 + 9	
25	Return Strip num = FPGA_ID*16 + 9	
4	Direct Strip num = FPGA_ID*16 + 11	
27	Return Strip num = FPGA_ID*16 + 11	
2	Direct Strip num = FPGA_ID*16 + 13	
29	Return Strip num = FPGA_ID*16 + 13	
0	Direct Strip num = FPGA_ID*16 + 15	
31	Return Strip num = FPGA_ID*16 + 15	

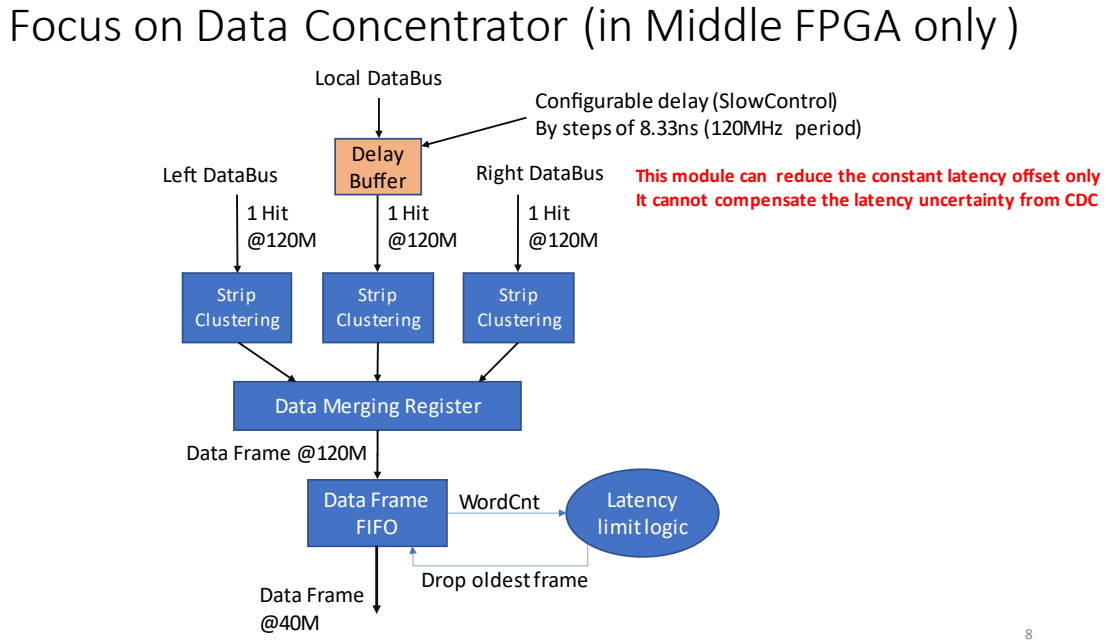
Time disparity control



6

Data Concentrator Module

The data concentrator is intended to compress and merge data provided by the data buses of the 3 FPGAs. It also produces the uplink GBT data frames and offers a mechanism of latency regulation based on the output activity. This module is only present in the Middle FPGA (the only one which communicates with the GBTx chip).



8

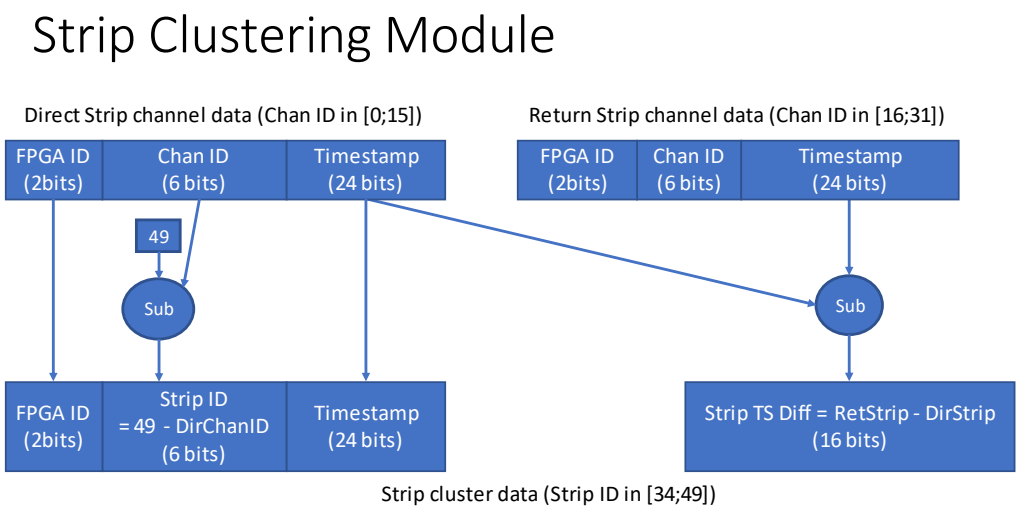
Middle FPGA delay buffer

A configurable delay buffer is instantiated on the Middle FPGA data bus (as this is the local bus) to compensate the latency due to inter-FPGA communication latency of the other buses. This is

implemented to reduce the amount of buffering resources needed by the backend to resynchronize flows from the different FPGAs.

Strip Clustering

A strip clustering module is present on each data bus, its goal is to try to compress channel data issued from the same strip.



10

From the pair of channel data, the timestamp difference is processed and forwarded on 16bits along with the data corresponding to the Direct Strip channel (the timestamp from the return strip channel data is dropped as it can be fully reconstructed). The channel ID is also re-numbered to produce a strip ID.

The “Remove single data” register allows to force clustering and drop isolated channel data.

It is unadvised to enable the “Remove single data” for a FPGA where the pair filtering is not enabled (if the pair filtering is not enabled for one strip, related data have a lot of chance to be dropped at the concentrator level).

Data from Resync and BC0 channels are not affected by this module and are just recorded in the data merging register as soon as they come.

Data merging, Frame creation and latency regulation

Data from the 3 buses are then merged to produce output data frame. This data merging is designed to keep as much as possible a good balance between the amount of data from the 3 FPGAs.

The output frame is then recorded in the frame queue. As this queue is written at data bus frequency (120MHz) and read at GBT frame frequency (40MHz), the number of pending frames can grow up to a limit (defined by the “Output Frame Queue Maximum Size” register). When the limit is hit, oldest frames in the queue are dropped in order to keep a hard latency limit to the data concentrator module.

GBT-SCA features

The GBT-SCA is an ASIC which provides the slow control management of many of the FEB devices. It is directly linked to the GBTX with a bidirectional 80Mb/s Elink.

https://espace.cern.ch/GBT-Project/GBT-SCA/Manuals/GBT-SCA_Manual_2019.002.pdf

GPIO

SCA owns 32 General Purpose digital IO lines (GPIO). Each line can be individually programmed as input or output or in a tri-state mode.

The table below shows the connections between the SCA GPIOs and the other devices for the FEBv2r3:

GPIO index	SCA Port Direction	GPIO ID	Source/Destination
0	input	CRC_ERROR_LEFT	From FPGAs (via lvl shifter)
1	input	CRC_ERROR_MIDDLE	
2	input	CRC_ERROR_RIGHT	
3	output	NCONFIG_LEFT	To FPGAs (via lvl shifter)
4	output	NCONFIG_MIDDLE	
5	output	NCONFIG_RIGHT	
6	input	INIT_DONE_LEFT	From FPGAs (via lvl shifter)
7	input	INIT_DONE_MIDDLE	
8	input	INIT_DONE_RIGHT	
9	output	ENABLE_POWER_FPGAS	To Power Supplies
10	output	NOT_USED	To FPGAs
11	input	FAULT_4V	From Overcurrent protection
12	input	FAULT_2V	
13	input	ALERT_LEFT	From Temperature sensors
14	input	ALERT_MIDDLE	
15	input	ALERT_RIGHT	
16	input	ALERT_2V	
17	input	ALERT_4V	
18	output	SCA_JTAG_SELECT#	To JTAG MUX
19	input	SPARE_IO_LEFT_2	From Left FPGA
20	input	SPARE_IO_LEFT_1	
21	output	SOFT_RESET_LEFT	To Left FPGA
22	output	SHDN_2V	To Overcurrent protection
23	output	SCA_SPI_EN	To SPI MUX
24	input	SPARE_IO_MIDDLE_2	From Middle FPGA
25	input	SPARE_IO_MIDDLE_1	
26	output	SOFT_RESET_MIDDLE	To Middle FPGA
27	output	SHDN_4V	To Overcurrent protection
28	input	NOT_USED	Open
29	input	SPARE_IO_RIGHT_2	From Right FPGA
30	input	SPARE_IO_RIGHT_1	
31	output	SOFT_RESET_RIGHT	To Right FPGA

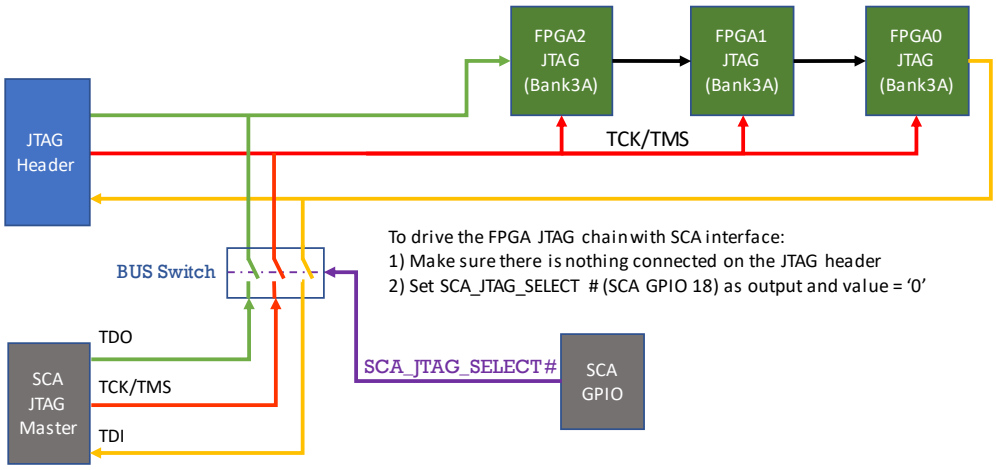
To avoid power supply related problems, it is highly advised to always set the 3 SOFT_RESET high before changing the values of N_CONFIG, ENABLE_POWER_FPGAS, SHDN_2V and SHDN_4V outputs.

JTAG chain

The 3 FPGAs of the board belong to a JTAG chain that can be accessed either by the JTAG header (a compatible INTEL FPGA programmer is needed) or through the GBT-SCA JTAG interface.

To make the SCA JTAG interface able to access to the chain, the corresponding SCA GPIO output has to be set LOW.

FEBv2r3 JTAG chain (simplified block diagram)



This JTAG chain can be used as an alternative way to remotely change the firmware located in the flash memories.

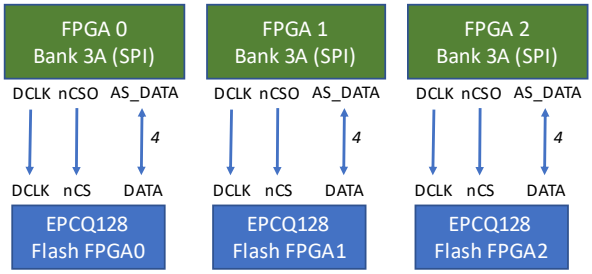
SPI Interface

Each FPGA has its own flash memory (ref: EPCQ128ASI16N) which stores the firmware loaded during a FPGA configuration sequence.

These memories can be accessed through the SPI Interface of the GBT-SCA, this is the only way to update the firmware when the FPGAs are not powered.

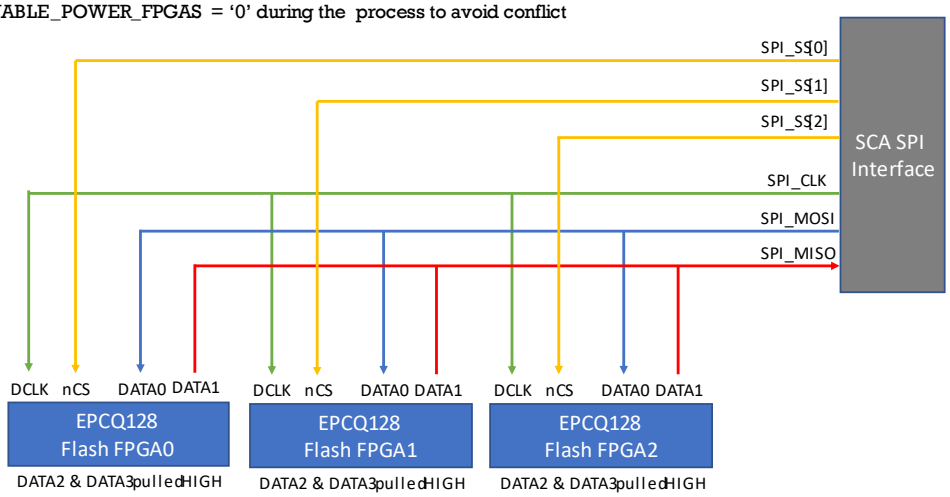
FEBv2r3 Flash SPI interface (simplifiedblock diagram)

When SCA_SPI_EN = '0' or 'Z' (default)
Each FPGA can write and read its own flash memory.



FEBv2r3 Flash SPI interface (simplifiedblock diagram)

When SCA_SPI_EN = '1' (SCA to Flash connection)
The flash memories can be accessed through the SCA SPI Interface
Make sure to set ENABLE_POWER_FPGAS = '0' during the process to avoid conflict



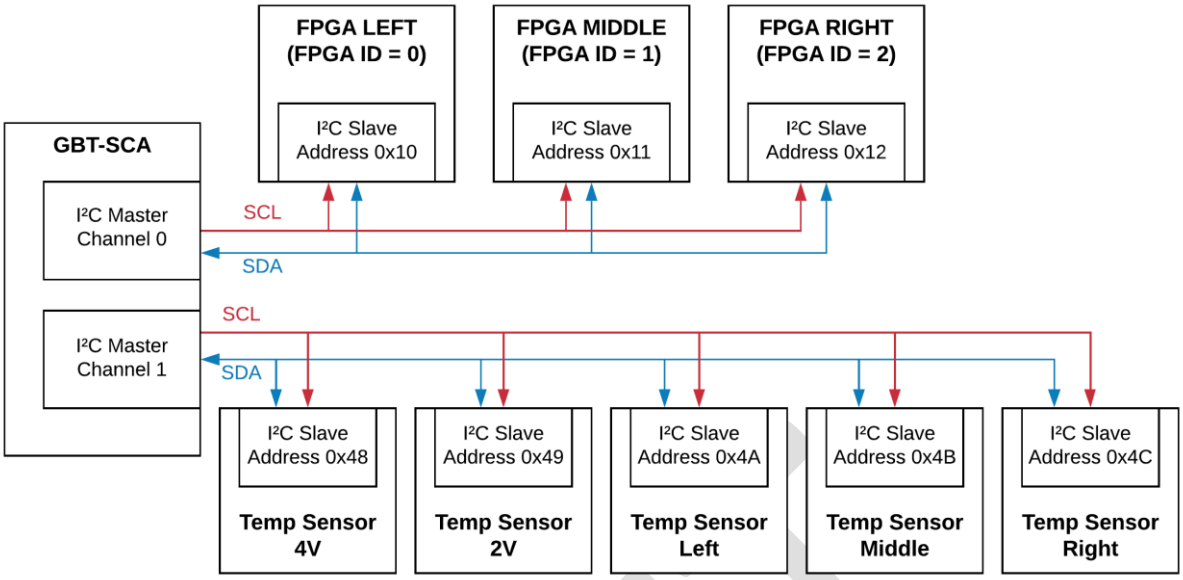
It is possible to write the same thing simultaneously in the 3 flash memories (in that case set the 3 SCA SPI_SS to '0'), but the content of only one flash can be read at a given time (keep the other SPI_SS to '1').

I²C Buses

SCA hosts 16 independent I²C Masters. On the FEB, only the 2 first buses are used:

- I²C channel 0 to communicate with the three FPGAs
- I²C channel 1 to access the different temperature sensors of the board

The following figure presents the topology of the I²C buses:



I²C FPGAs Bus

This bus allows to control FPGAs features that cannot be accessible with the GBT frame: eLinks loopback, word alignment (bitslip) configuration and pattern injection.

The I²C slaves instantiated in the FPGAs host 8bits registers addressed on 8bits.

Register Address		Register Name	Register range	Description	Access	Default Value
0	0x00	Pattern injection CTRL	[0]	'1' -> Elinks loopback enable	RW	0x00
			[1]	'1' -> Uplink debug pattern enable		
			[2]	'1' -> Debug pattern toggle enable		
			[3]	'1' -> TDC bus pattern injection enable		
			[4]	'1' -> Slow control bus pattern injection enable		
			[7:5]	IGNORED		
1	0x01	Calibration patterns	[7:0]	Uplink Pattern (eLink 0)	RW	0xAB
..	...		[7:0]	Uplink Pattern (eLink 1 -> 12)		0xAB
14	0x0E		[7:0]	Uplink Pattern (eLink 13)		0xAB
15	0x0F	Auto Word Alignment request	[0]	'1' -> Auto Word alignment request	RW	0x00
			[7:1]	IGNORED		
16	0x10	Manual Word Alignment bank 7A	[2:0]	Rx bitslip value for eLinks connected to FPGA bank 7A	RW	0x13
			[3]	IGNORED		
			[4]	'1' -> Use the Rx bitslip value (bank 7A)		
			[7:5]	IGNORED		
17	0x11	Manual Word Alignment bank 4A	[2:0]	Rx bitslip value for eLinks connected to FPGA bank 4A	RW	0x13
			[3]	IGNORED		
			[4]	'1' -> Use the Rx bitslip value (bank 4A)		
			[7:5]	IGNORED		
18	0x12	TxWordAlignmentConfig	[2:0]	Tx bitslip value for every uplink eLinks	RW	0x04
			[7:3]	IGNORED		
19	0x13	Gxb Force Byte alignment	[0]	'1' -> Force the Gxb transceivers in locked mode	RW	0x00
			[7:1]	IGNORED		
...	..	NOT USED				
32	0x20	Auto Word Alignment Results	[2:0]	Calculated downlink bitslip for bank 4A	RO	N/A
			[3]	'1' -> Auto Word alignment success for bank 4A		
			[6:4]	Calculated downlink bitslip for bank 7A		
			[7]	'1' -> Auto Word alignment success for bank 7A		

To read a register (1 word I²C write transaction followed by 1 word I²C read transaction):

I²C address, Reg Address

To write a register (2 words I²C write transaction):

I²C address, Reg Address, data byte

I²C Temperature Sensors Bus

This bus is used to configure and read the temperature from the LM75 temperature sensors of the board.

https://www.mouser.com/datasheet/2/282/snis153a-123211.pdf

This component owns an internal pointer, which is used for addressing read transaction. This pointer value is refreshed by an I²C write transaction. The internal pointer default value (at power on) is 0.

To change only internal pointer value (1 word I²C write transaction):

I²C address, Reg Address

To write an 8 bits register (2 words I²C write transaction):

I²C address, Reg Address, data byte

To write a 16 bits register (3 words I²C write transaction):

I²C address, Reg Address, most significant data byte, less significant data byte

---

To read an 8 bits register or a 16 bits register, the master must generate respectively a 1 word I²C read transaction or a 2 words I²C read transaction. The return byte(s) is(are) the content of the register pointed by the internal pointer.

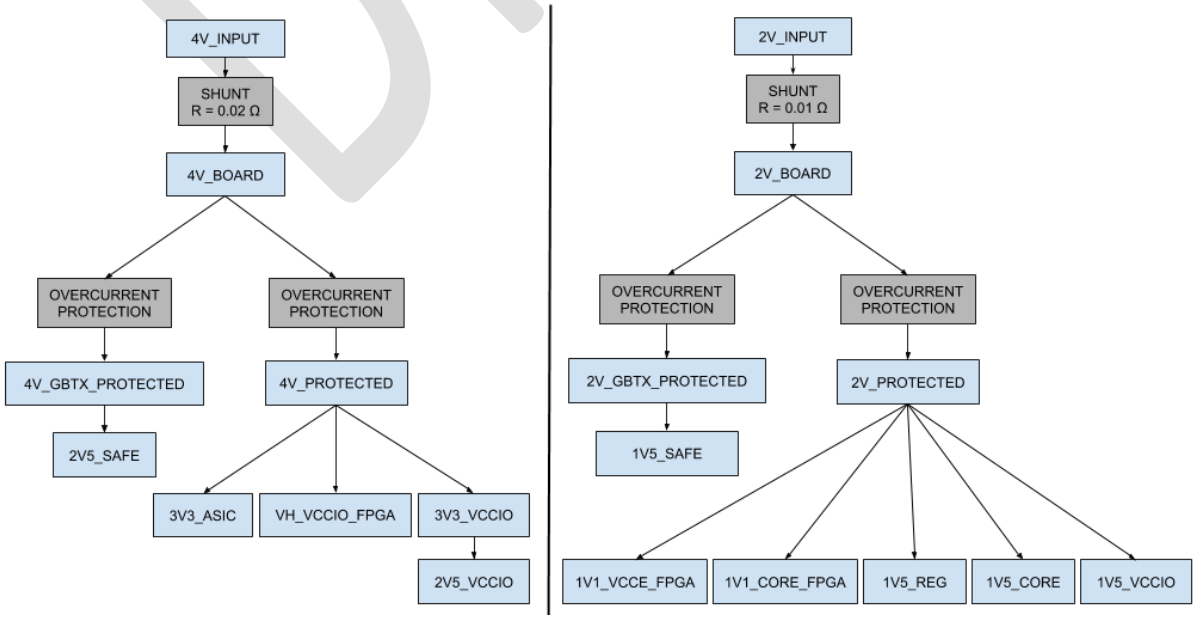
Register Address	Register Name	Access	Description	Default Value
0x00	Temperature	RO	Read the 9 most significant bits ([15:7]) to get the temperature. The 7 less significant bits ([6:0]) are undefined.	N/A
0x01	Configuration	RW	**WARNING : This Register is ONLY 1 BYTE LONG !!! Read and write transactions requires ONLY 1 DATA BYTE**	0x00
0x02	T_{HYST} set point	RW	Write the T_{HYST} value in the 9 most significant bits ([15:7]). The 7 less significant bits ([6:0]) are ignored.	75°C
0x03	T_{OS} set point	RW	Write the T_{OS} value in the 9 most significant bits ([15:7]). The 7 less significant bits ([6:0]) are ignored.	80°C

ADC

To monitor voltage and current of the different power supplies and regulators of the board.

ADC index	ADC ID	Source	ADC input equation
0	RSSI_ADC	From VTRX	
1	1V5_VCCIO_ADC	From VCCIO Regulators (via voltage divider)	= 1/2 * 1V5_VCCIO
2	2V5_VCCIO_ADC		= 1/3 * 2V5_VCCIO
3	3V3_VCCIO_ADC		= 10/49 * 3V3_VCCIO
4	1V5_SAFE_ADC	From VTRX/GBTX Regulators (via voltage divider)	= 1/2 * 1V5_SAFE
5	2V5_SAFE_ADC		= 1/3 * 2V5_SAFE
6	1V1_CORE_LEFT_ADC	From FPGA CORE VCC Regulators	= 1/2 * 1V1_CORE_LEFT
7	1V1_CORE_MIDDLE_ADC		= 1/2 * 1V1_CORE_MIDDLE
8	1V1_CORE_RIGHT_ADC		= 1/2 * 1V1_CORE_RIGHT
9	1V1_VCCE_LEFT_ADC	From FPGA GXB Regulators	= 1/2 * 1V1_VCCE_LEFT
10	1V1_VCCE_MIDDLE_ADC		= 1/2 * 1V1_VCCE_MIDDLE
11	1V1_VCCE_RIGHT_ADC		= 1/2 * 1V1_VCCE_RIGHT
12	2V_CURRENT_ADC	Power input (via operational amplifier)	= 10 * (2V_INPUT - 2V_BOARD)
13	4V_CURRENT_ADC		= 10 * (4V_INPUT - 4V_BOARD)
14	FPGA_CORE_CURRENT_LEFT_ADC	From FPGA Regulators (via operational amplifier)	= 10 * (1V5_REG_LEFT - 1V5_CORE_LEFT)
15	FPGA_CORE_CURRENT_MIDDLE_ADC		= 10 * (1V5_REG_MIDDLE - 1V5_CORE_MIDDLE)
16	FPGA_CORE_CURRENT_RIGHT_ADC		= 10 * (1V5_REG_RIGHT - 1V5_CORE_RIGHT)
17	VH_VCCIO_LEFT_ADC	From FPGA VCCIO Regulators (via voltage divider)	= 1/3 * VH_VCCIO_LEFT
18	VH_VCCIO_MIDDLE_ADC		= 1/3 * VH_VCCIO_MIDDLE
19	VH_VCCIO_RIGHT_ADC		= 1/3 * VH_VCCIO_RIGHT
20	2V_BOARD_ADC	Power input (via operational amplifier)	= 1/3 * 2V_BOARD
21	4V_BOARD_ADC		= 10/49 * 4V_BOARD
22	**NOT USED**		
23			
24			
25			
26			
27			
28			
29			
30			
31			

Board power management recap



Clocks

To perform their tasks, the three FPGAs must receive different external clocks and derive other clocks internally using a PLL component.

External clocks

Main system clock - 40MHz

This clock is related to the bunch-crossing clock of the detector and is provided to the FPGA by the GBTx ASIC. In the firmware, this is the clock which is used as input of a pll to create all the internal clocks.

TDC calibration clock – 11MHz

This clock is required by the TDC in order to calibrate itself, the LUT building procedure is made possible by this clock.

eLink clocks - 40Mhz (Middle FPGA only)

These clocks are provided by the GBTx ASIC in order to help the serialization/deserialization of eLinks data in the middle FPGA. Using a 40MHz instead of a 320MHz allows to give information of the word reconstruction phase. The deserialization clocks (to sample the 320Mbps downlink eLinks data) and the serialization clocks (to send the 320Mbps uplink eLinks data) are derived from these 40MHz by a PLL in the SerDes module of the middle FPGA.

Internal clocks

All of these internal clocks are derived from the main 40MHz clock inside each of the 3 FPGAs.

TDC Clock - 400MHz

This is the operating clock of the TDC, it corresponds to the coarse time resolution of the channels.

Main Slow control and Data bus clock - 120MHz

This is the data concentration clock, its frequency allows to reach the maximum uplink GBT frame data bandwidth (capacity of 3 TDC data within the 40MHz GBT frame). This clock is also used in the slow control master/slave communication within the three FPGAs.

GXB reference clock - 120MHz

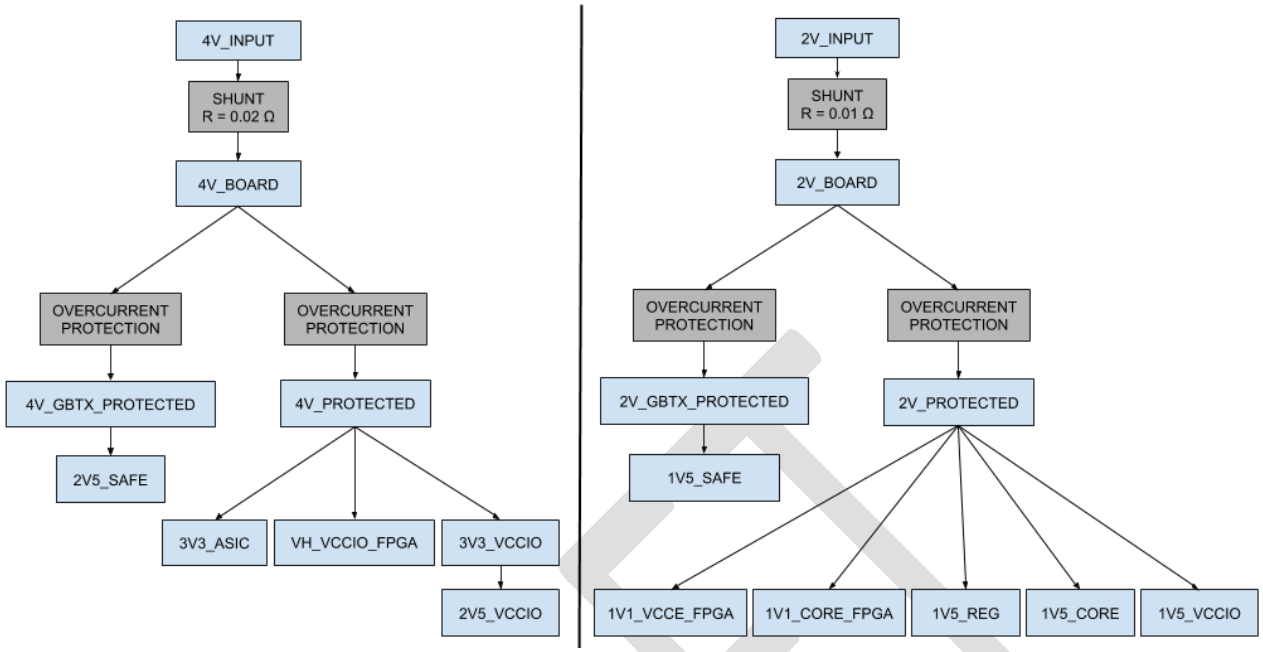
This is the clock used within the high-speed transceivers required for the communication between the different FPGAs.

Fast control clock - 80MHz (Middle FPGA only)

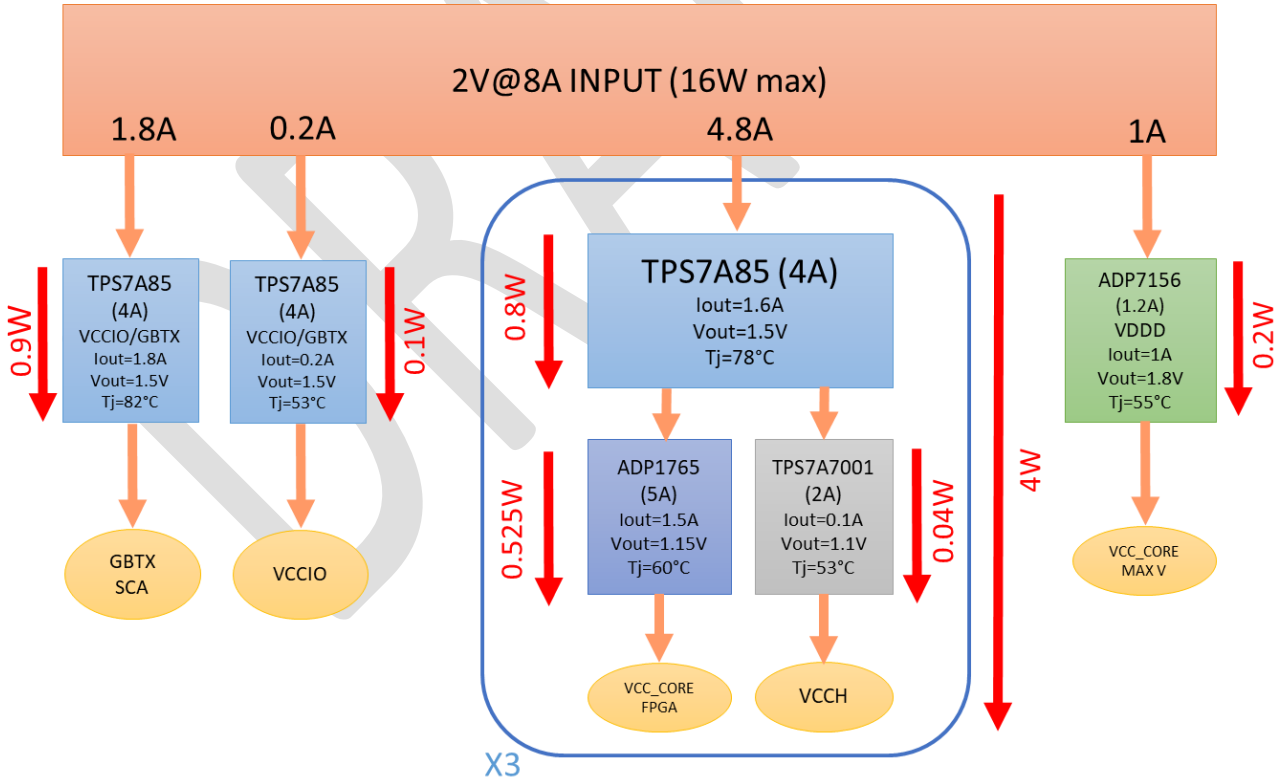
This is the clock used to generate the Resync of BC0 pulses upon reception of the corresponding fast control command in a downlink GBT Frame. A mechanism is here to make the generated pulse only lasts one 80MHz clock period. This allows to correctly generate pulses when many fast control command are send is consecutive GBT frames.

FEB power management

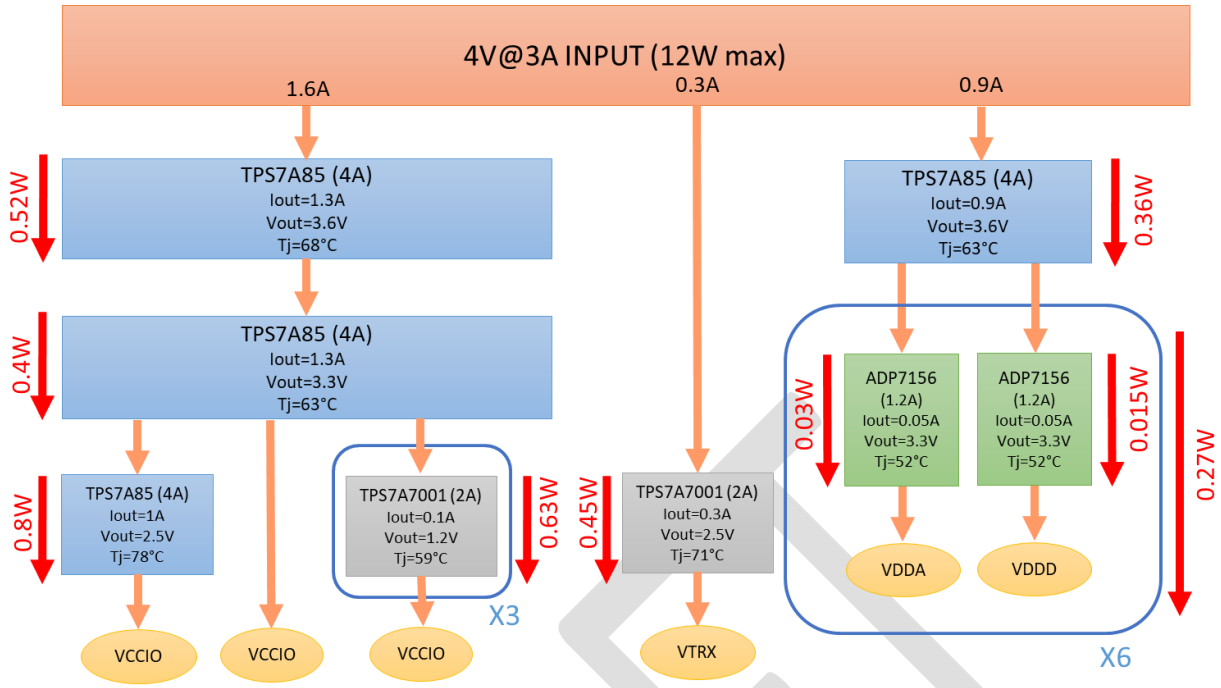
Board power management overview



2V power path details



4V power path details



Power supply estimation

POWER SUPPLY	2V					4V		
COMPONENT / VOLTAGE	Digital	Digital	Digital	Digital	Digital	Digital	Digital	Analog
	1.1V	1.15V	1.2V	1.5V	1.8V	2.5V	3.3V	3.3V
VTRX	0	0	0	0	0	0.3	0	0
GBT-SCA	0	0	0	0.1	0	0	0	0
GBTX	0	0	0	1.56	0	0	0	0
GBTX_LEDS (4 leds)	0	0	0	0	0	0.04	0	0
MAX V	0	0	0	0.05	0.5	0.2	0	0
FPGA 1	0.1	1.5	0.05	0	0	0.2	0.05	0
FPGA 2	0.1	1.5	0.05	0.2	0	0.2	0.05	0
FPGA 3	0.1	1.5	0.05	0	0	0.2	0.05	0
FPGAs_LEDS (9 leds)	0	0	0	0	0	0	0.09	0
ASIC 1	0	0	0.05	0	0	0	0.05	0.06
ASIC 2	0	0	0.05	0	0	0	0.05	0.06
ASIC 3	0	0	0.05	0	0	0	0.05	0.06
ASIC 4	0	0	0.05	0	0	0	0.05	0.06
ASIC 5	0	0	0.05	0	0	0	0.05	0.06
ASIC 6	0	0	0.05	0	0	0	0.05	0.06
KC3225A-100MHz	0	0	0	0	0	0.02	0	0
ASE2-40MHz	0	0	0	0	0	0.025	0	0
EG2121CA-200MHz	0	0	0	0	0	0.05	0	0
EG2121CA-400MHz	0	0	0	0	0	0.05	0	0
SN65LVDS108 (2)	0	0	0	0	0	0	0.2	0
TXB0104	0	0	0	0.1	0	0.1	0	0
S29GL01GS	0	0	0	0	0	0.1	0.1	0
CURRENT (A)	0.3	4.5	0.45	2.01	0.5	1.485	0.84	0.36
POWER (W)	0.33	5.175	0.54	3.015	0.9	3.7125	2.772	1.188
	POWER	CURRENT						
POWER SUPPLY 2V	9.96	7.76						
POWER SUPPLY 4V	7.6725	2.685						

FEBv2_r3 Data channel mapping

FEBv2r3 Strip/PETIROC/TDC channel mapping

FPGA LEFT (0)

	PETIROC TOP (Direct Strip)			PETIROC BOTTOM (Return Strip)	
Strip n°	PETIROC Channel	FPGA TDC Channel	RStrip n°	PETIROC Channel	FPGA TDC Channel
15	5	0	15	26	31
14	6	1	14	25	30
13	7	2	13	24	29
12	8	3	12	23	28
11	9	4	11	22	27
10	10	5	10	21	26
9	11	6	9	20	25
8	12	7	8	19	24
7	19	8	7	12	23
6	20	9	6	11	22
5	21	10	5	10	21
4	22	11	4	9	20
3	23	12	3	8	19
2	24	13	2	7	18
1	25	14	1	6	17
0	26	15	0	5	16

FPGA MIDDLE (1)

	PETIROC TOP (Direct Strip)			PETIROC BOTTOM (Return Strip)	
Strip n°	PETIROC Channel	FPGA TDC Channel	RStrip n°	PETIROC Channel	FPGA TDC Channel
31	5	0	31	26	31
30	6	1	30	25	30
29	7	2	29	24	29
28	8	3	28	23	28
27	9	4	27	22	27
26	10	5	26	21	26
25	11	6	25	20	25
24	12	7	24	19	24
23	19	8	23	12	23
22	20	9	22	11	22
21	21	10	21	10	21
20	22	11	20	9	20
19	23	12	19	8	19
18	24	13	18	7	18
17	25	14	17	6	17
16	26	15	16	5	16

FPGA RIGHT (2)

	PETIROC TOP (Direct Strip)			PETIROC BOTTOM (Return Strip)	
Strip n°	PETIROC Channel	FPGA TDC Channel	RStrip n°	PETIROC Channel	FPGA TDC Channel
47	5	0	47	26	31
46	6	1	46	25	30
45	7	2	45	24	29
44	8	3	44	23	28
43	9	4	43	22	27
42	10	5	42	21	26
41	11	6	41	20	25
40	12	7	40	19	24
39	19	8	39	12	23
38	20	9	38	11	22
37	21	10	37	10	21
36	22	11	36	9	20
35	23	12	35	8	19
34	24	13	34	7	18
33	25	14	33	6	17
32	26	15	32	5	16

For each FPGA:	TDC Channel 32	BC0
	TDC Channel 33	Resync

PETIROC configuration reference

In the following tab, you can find information on parameters to configure the PETIROC slow control registers with a proposed (tested) value. The values that need to be controlled/changed accordingly to the wanted FEBv2_r3 utilization are highlighted in **green**.

Name	Bit index	Size	Order	Proposed value
mask_discri_charge_ch0	0	1		1
mask_discri_charge_ch1	1	1		1
mask_discri_charge_ch2	2	1		1
mask_discri_charge_ch3	3	1		1
mask_discri_charge_ch4	4	1		1
mask_discri_charge_ch5	5	1		1
mask_discri_charge_ch6	6	1		1
mask_discri_charge_ch7	7	1		1
mask_discri_charge_ch8	8	1		1
mask_discri_charge_ch9	9	1		1
mask_discri_charge_ch10	10	1		1
mask_discri_charge_ch11	11	1		1
mask_discri_charge_ch12	12	1		1
mask_discri_charge_ch13	13	1		1
mask_discri_charge_ch14	14	1		1
mask_discri_charge_ch15	15	1		1
mask_discri_charge_ch16	16	1		1
mask_discri_charge_ch17	17	1		1
mask_discri_charge_ch18	18	1		1
mask_discri_charge_ch19	19	1		1
mask_discri_charge_ch20	20	1		1
mask_discri_charge_ch21	21	1		1
mask_discri_charge_ch22	22	1		1
mask_discri_charge_ch23	23	1		1
mask_discri_charge_ch24	24	1		1
mask_discri_charge_ch25	25	1		1
mask_discri_charge_ch26	26	1		1
mask_discri_charge_ch27	27	1		1
mask_discri_charge_ch28	28	1		1
mask_discri_charge_ch29	29	1		1
mask_discri_charge_ch30	30	1		1
mask_discri_charge_ch31	31	1		1
input_dac_ch0	32	8	LSB first	0x80
cmd_input_dac_ch0	40	1		1
input_dac_ch1	41	8	LSB first	0x80
cmd_input_dac_ch1	49	1		1
input_dac_ch2	50	8	LSB first	0x80
cmd_input_dac_ch2	58	1		1
input_dac_ch3	59	8	LSB first	0x80
cmd_input_dac_ch3	67	1		1

input_dac_ch4	68	8	LSB first	0x80
cmd_input_dac_ch4	76	1		1
input_dac_ch5	77	8	LSB first	0x80
cmd_input_dac_ch5	85	1		1
input_dac_ch6	86	8	LSB first	0x80
cmd_input_dac_ch6	94	1		1
input_dac_ch7	95	8	LSB first	0x80
cmd_input_dac_ch7	103	1		1
input_dac_ch8	104	8	LSB first	0x80
cmd_input_dac_ch8	112	1		1
input_dac_ch9	113	8	LSB first	0x80
cmd_input_dac_ch9	121	1		1
input_dac_ch10	122	8	LSB first	0x80
cmd_input_dac_ch10	130	1		1
input_dac_ch11	131	8	LSB first	0x80
cmd_input_dac_ch11	139	1		1
input_dac_ch12	140	8	LSB first	0x80
cmd_input_dac_ch12	148	1		1
input_dac_ch13	149	8	LSB first	0x80
cmd_input_dac_ch13	157	1		1
input_dac_ch14	158	8	LSB first	0x80
cmd_input_dac_ch14	166	1		1
input_dac_ch15	167	8	LSB first	0x80
cmd_input_dac_ch15	175	1		1
input_dac_ch16	176	8	LSB first	0x80
cmd_input_dac_ch16	184	1		1
input_dac_ch17	185	8	LSB first	0x80
cmd_input_dac_ch17	193	1		1
input_dac_ch18	194	8	LSB first	0x80
cmd_input_dac_ch18	202	1		1
input_dac_ch19	203	8	LSB first	0x80
cmd_input_dac_ch19	211	1		1
input_dac_ch20	212	8	LSB first	0x80
cmd_input_dac_ch20	220	1		1
input_dac_ch21	221	8	LSB first	0x80
cmd_input_dac_ch21	229	1		1
input_dac_ch22	230	8	LSB first	0x80
cmd_input_dac_ch22	238	1		1
input_dac_ch23	239	8	LSB first	0x80
cmd_input_dac_ch23	247	1		1
input_dac_ch24	248	8	LSB first	0x80
cmd_input_dac_ch24	256	1		1
input_dac_ch25	257	8	LSB first	0x80
cmd_input_dac_ch25	265	1		1
input_dac_ch26	266	8	LSB first	0x80
cmd_input_dac_ch26	274	1		1

input_dac_ch27	275	8	LSB first	0x80
cmd_input_dac_ch27	283	1		1
input_dac_ch28	284	8	LSB first	0x80
cmd_input_dac_ch28	292	1		1
input_dac_ch29	293	8	LSB first	0x80
cmd_input_dac_ch29	301	1		1
input_dac_ch30	302	8	LSB first	0x80
cmd_input_dac_ch30	310	1		1
input_dac_ch31	311	8	LSB first	0x80
cmd_input_dac_ch31	319	1		1
input_dac_ch_dummy	320	8	LSB first	0x80
mask_discri_time_ch0	328	1		1
mask_discri_time_ch1	329	1		1
mask_discri_time_ch2	330	1		1
mask_discri_time_ch3	331	1		1
mask_discri_time_ch4	332	1		1
mask_discri_time_ch5	333	1		1
mask_discri_time_ch6	334	1		1
mask_discri_time_ch7	335	1		1
mask_discri_time_ch8	336	1		1
mask_discri_time_ch9	337	1		1
mask_discri_time_ch10	338	1		1
mask_discri_time_ch11	339	1		1
mask_discri_time_ch12	340	1		1
mask_discri_time_ch13	341	1		1
mask_discri_time_ch14	342	1		1
mask_discri_time_ch15	343	1		1
mask_discri_time_ch16	344	1		1
mask_discri_time_ch17	345	1		1
mask_discri_time_ch18	346	1		1
mask_discri_time_ch19	347	1		1
mask_discri_time_ch20	348	1		1
mask_discri_time_ch21	349	1		1
mask_discri_time_ch22	350	1		1
mask_discri_time_ch23	351	1		1
mask_discri_time_ch24	352	1		1
mask_discri_time_ch25	353	1		1
mask_discri_time_ch26	354	1		1
mask_discri_time_ch27	355	1		1
mask_discri_time_ch28	356	1		1
mask_discri_time_ch29	357	1		1
mask_discri_time_ch30	358	1		1
mask_discri_time_ch31	359	1		1
6b_dac_ch0	360	6	LSB first	0x01
6b_dac_ch1	366	6	LSB first	0x01
6b_dac_ch2	372	6	LSB first	0x01

6b_dac_ch3	378	6	LSB first	0x01
6b_dac_ch4	384	6	LSB first	0x01
6b_dac_ch5	390	6	LSB first	0x01
6b_dac_ch6	396	6	LSB first	0x01
6b_dac_ch7	402	6	LSB first	0x01
6b_dac_ch8	408	6	LSB first	0x01
6b_dac_ch9	414	6	LSB first	0x01
6b_dac_ch10	420	6	LSB first	0x01
6b_dac_ch11	426	6	LSB first	0x01
6b_dac_ch12	432	6	LSB first	0x01
6b_dac_ch13	438	6	LSB first	0x01
6b_dac_ch14	444	6	LSB first	0x01
6b_dac_ch15	450	6	LSB first	0x01
6b_dac_ch16	456	6	LSB first	0x01
6b_dac_ch17	462	6	LSB first	0x01
6b_dac_ch18	468	6	LSB first	0x01
6b_dac_ch19	474	6	LSB first	0x01
6b_dac_ch20	480	6	LSB first	0x01
6b_dac_ch21	486	6	LSB first	0x01
6b_dac_ch22	492	6	LSB first	0x01
6b_dac_ch23	498	6	LSB first	0x01
6b_dac_ch24	504	6	LSB first	0x01
6b_dac_ch25	510	6	LSB first	0x01
6b_dac_ch26	516	6	LSB first	0x01
6b_dac_ch27	522	6	LSB first	0x01
6b_dac_ch28	528	6	LSB first	0x01
6b_dac_ch29	534	6	LSB first	0x01
6b_dac_ch30	540	6	LSB first	0x01
6b_dac_ch31	546	6	LSB first	0x01
EN_10bits_DAC	552	1		1
PP_10bits_DAC	553	1		1
10b_dac_vth_discri_charge	554	10	MSB first	0x000
10b_dac_vth_discri_time	564	10	MSB first	0x1F4
EN_ADC	574	1		0
PP_ADC	575	1		0
sel_startb_ramp_ADC_ext	576	1		0
usebcompensation	577	1		0
EN_bias_DAC_delay	578	1		1
PP_bias_DAC_delay	579	1		1
EN_bias_ramp_delay	580	1		0
PP_bias_ramp_delay	581	1		0
8b_dac_delay	582	8	LSB first	0x00
EN_discri_delay	590	1		1
PP_discri_delay	591	1		1
PP_temp_sensor	592	1		0
EN_temp_sensor	593	1		0

EN_bias_pa	594	1		1
PP_bias_pa	595	1		1
EN_bias_discri	596	1		1
PP_bias_discri	597	1		1
cmd_polarity	598	1		0
latch_discri	599	1		1
EN_bias_6b_dac	600	1		1
PP_bias_6b_dac	601	1		1
EN_bias_tdc	602	1		0
PP_bias_tdc	603	1		0
ON_OFF_input_dac	604	1		1
EN_bias_charge	605	1		0
PP_bias_charge	606	1		0
cf_100fF	607	1		0
cf_200fF	608	1		0
cf_2_5pF	609	1		0
cf_1_25pF	610	1		0
EN_bias_sca	611	1		0
PP_bias_sca	612	1		0
EN_bias_discri_charge	613	1		0
PP_bias_discri_charge	614	1		0
EN_bias_discri_adc_time	615	1		0
PP_bias_discri_adc_time	616	1		0
EN_bias_discri_adc_charge	617	1		0
PP_bias_discri_adc_charge	618	1		0
DIS_razchn_int	619	1		1
DIS_razchn_ext	620	1		0
SEL_80M	621	1		0
EN_80M	622	1		0
EN_slow_lvds_rec	623	1		1
PP_slow_lvds_rec	624	1		1
EN_fast_lvds_rec	625	1		1
PP_fast_lvds_rec	626	1		0
EN_transmitter	627	1		0
PP_transmitter	628	1		0
ON_OFF_1mA	629	1		1
ON_OFF_2mA	630	1		1
NC1	631	1		0
ON_OFF_ota_mux	632	1		0
ON_OFF_ota_probe	633	1		0
DIS_trig_mux	634	1		1
EN_NOR32_time	635	1		0
EN_NOR32_charge	636	1		0
DIS_triggers	637	1		0
EN_dout_oc	638	1		0
EN_transmit	639	1		0

PA_Ccomp<0>	640	1		0	2B & 2C only
PA_Ccomp<1>	641	1		0	
PA_Ccomp<2>	642	1		0	
PA_Ccomp<3>	643	1		1	
NC2	644	1		1	
NC3	645	1		0	
NC4	646	1		0	
Choice_trigger_out	647	1		0	
Delay_reset_trigger	648	4	LSB first	0x0	2C only
NC5	652	1		0	
NC6	653	1		0	
NC7	654	1		0	
En_reset_trigger_delay	655	1		0	
Delay_reset_ToT	656	4	LSB first	0x0	
NC8	660	1		0	
NC9	661	1		0	
NC10	662	1		0	
EN_reset_ToT_delay	663	1		0	

APPENDIX: Outdated material (removed features, previous board revisions…)

Quick start guide

WARNING: Section refers to an old revision of the FEB (FEBv2r2)

This guide is a checklist that provides the shortest path to get the FEB functional (every feature accessible). It is assumed you already have a backend board implementing a well-configured GBT_FPGA module and this system is capable of producing GBT frames and interpreting GBT frames accordingly to the format specified in the FEB specification document.

Installation

Plug 4 power cables between a power supply and the FEB (2V/GND and 4V/GND).

Plug the optical link between the backend board that hosts the GBT FPGA and the FEB.

Tune and enable the power supply.

Depending on your setup, it can be useful to compensate the voltage drop caused by power wires. If you are using power wires longer than a meter, it is highly recommended to use a regulated power supply with sense wires (the nominal current for the input 2V supply of the FEB is higher than 6A, this causes a large voltage drop due to the wire resistance).

If the board is correctly powered, the LEDs PGOOD_SAFE and RX_READY are ON.

If the GBT_FPGA module of the backend board is correctly working, the LED TX_READY is ON.

As the GBTx present on the board has a fused configuration, the GBT link is ready to communicate with the SCA.

Configuring SCA

To perform this configuration, the SCA must be accessed using the EC (External Control) field of the GBT frame. To generate conveniently the external control bits of the frame, it is advised to implement a GBT-SC module (CERN development) in the backend, next to the GBT FPGA module. For more details, check SCA documentation.

SCA Ctrl reg initialization

The goal of this step is to setup the SCA channels used to monitor and control FEB features (GPIOs, ADCs and the two I^2C buses).

SCA channels enabling:

COMMAND	FUNCTION	TYPE	CH	TRN	CMD/ER	D[31:24]	D[23:16]	D[15:8]	D[7:0]
CTRL_R_ID	Read the Chip ID	TX:	0x14	N	V2-> 0xD1 V1-> 0x91	-	-	-	1
		RX:	0x14	N	0x00	-	ID[23:16]	ID[15:8]	ID[7:0]
CTRL _W_CRB	Write Control reg. B	TX:	0	N	0x02	VAL	-	-	-
		RX:	0	N	0x00	-	-	-	-
CTRL _W_CRC	Write Control reg. C	TX:	0	N	0x04	VAL	-	-	-
		RX:	0	N	0x00	-	-	-	-
CTRL _W_CRD	Write Control reg. D	TX:	0	N	0x06	VAL	-	-	-
		RX:	0	N	0x00	-	-	-	-
CTRL _R_CRB	Read Control reg. B	TX:	0	N	0x03	-	-	-	-
		RX:	0	N	0x00	VAL	-	-	-
CTRL _R_CRC	Read Control reg. C	TX:	0	N	0x05	-	-	-	-
		RX:	0	N	0x00	VAL	-	-	-
CTRL _R_CRD	Read Control reg. D	TX:	0	N	0x07	-	-	-	-
		RX:	0	N	0x00	VAL	-	-	-

Set the correct configuration for the three control registers of the SCA:

- CRB (CMD = 0x02) VAL = 0x1C
- CRC (CMD = 0x04) VAL = 0x00
- CRD (CMD = 0x06) VAL = 0x10

I²C bus configuration

COMMAND	FUNCTION	TYPE	CH	TRN	CMD/ER	D[31:24]	D[23:16]	D[15:8]	D[7:0]
I2C_W_CTRL	Write CONTROL register	TX:	I2cx	N	**0x30**	VALUE	--	--	--
		RX:	I2cx	N	flag	--	--	--	--

Configure the two I²C buses managed by the SCA:

- I²C Channel0 configuration (CH 0x03, CMD 0x30) VALUE = 0x88
- I²C Channel1 configuration (CH 0x04, CMD 0x30) VALUE = 0x88

The proposed value (0x88) stands for a “100kHz”, “nbytes=2”, “CMOS” configuration.

GPIOs direction configuration

COMMAND	FUNCTION	REQUEST AND REPLY FORMATS							
		TYPE	CH	TRN	LEN	CMD/ER	D[31:24]	D[23:16]	D[15:8]	D[7:0]
GPIO_W_DIRECTION	Write register DIRECTION	TX:	0x02	N	4	**0x20**	D[31:24]	D[23:16]	D[15:8]	D[7:0]
		RX:	0x02	N	2	Flag	--	--	--	--
GPIO_R_DIRECTION	Read register DIRECTION	TX:	0x02	N	1	**0x21**	--	--	--	--
		RX:	0x02	N	4	Flag	D[31:24]	D[23:16]	D[15:8]	D[7:0]

Configure the GPIO direction register:

- GPIO direction configuration (CH 0x02, CMD 0x20) D = 0x08 0x40 0x06 0x38

Power enabling

Once GPIO direction setting is done, SCA GPIOs defined as OUTPUT can be controlled, to enable power.

COMMAND	FUNCTION	REQUEST AND REPLY FORMATS							
		TYPE	CH	TRN	LEN	CMD/ER	D[31:24]	D[23:16]	D[15:8]	D[7:0]
GPIO_W_DATAOUT	Write register DATAOUT	TX:	0x02	N	4	**0x10**	D[31:24]	D[23:16]	D[15:8]	D[7:0]
		RX:	0x02	N	2	Flag	--	--	--	--
GPIO_R_DATAOUT	Read register DATAOUT	TX:	0x02	N	1	**0x11**	--	--	--	--
		RX:	0x02	N	4	Flag	D[31:24]	D[23:16]	D[15:8]	D[7:0]
GPIO_R_DATAIN	Read register DATAIN	TX:	0x02	N	1	**0x01**	--	--	--	--
		RX:	0x02	N	4	Flag	D[31:24]	D[23:16]	D[15:8]	D[7:0]

Using the DATAOUT register, bits corresponding to the 4V ENABLE (D[27]), 2V ENABLE (D[22]) and FPGA POWER ENABLE (D[9]) have to be set high.

- GPIO DATAOUT configuration (CH 0x02, CMD 0x10) D = 0x08 0x40 0x02 0x00

This turns on the LED PGOOD_CORE.

Important note:

As all the SCA GPIOs values are controlled by the same register, you have to send the full register content even if you want to change only one output.

To change only a subset of output values while keeping the others unchanged, you have 2 solutions:

- Read the register value, change only the bits you want and write the modified value.
- Keep a memory of the register value on the software side.

FPGA Configuration

FPGA firmware loading request is also done by controlling SCA GPIOs.

Activating the N_CONFIG pins of the FPGAs make them to load the firmware stored in the EEPROM.

GPIO mapping reminder:

- D[3] N_CONFIG_LEFT
- D[4] N_CONFIG_MIDDLE
- D[5] N_CONFIG_RIGHT
- D[21] SOFT_RESET_LEFT
- D[26] SOFT_RESET_MIDDLE
- D[31] SOFT_RESET_RIGHT

The first command requests the firmware loading for the 3 FPGAs, while the second releases the soft reset of the FPGAs:

- GPIO DATAOUT configuration (CH 0x02, CMD 0x10) D = 0x8C 0x60 0x02 0x38
- GPIO DATAOUT configuration (CH 0x02, CMD 0x10) D = 0x08 0x40 0x02 0x38

If the three onboard EEPROMs contain a firmware, the three FPGAs are now configured and accessible (green FPGA LEDs are blinking, 2 red FPGA LEDs are ON).

(Optional step) Load a new firmware in FPGAs/EEPROM

JTAG configuration through the Intel header

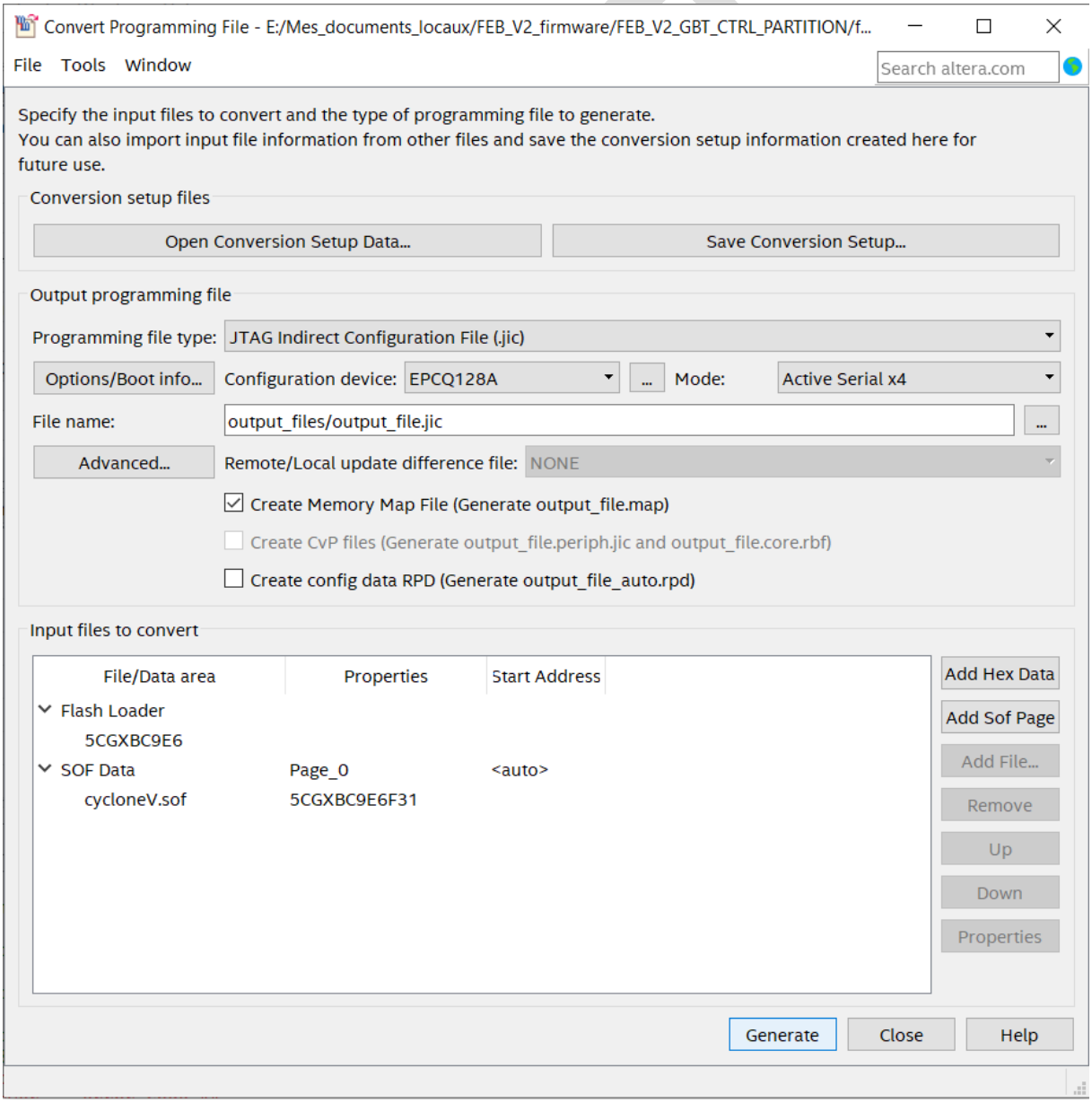
If you want to rewrite the EEPROM content (or directly the FPGA) with a new firmware, you can do it through the dedicated JTAG. This is the only way which allows to edit the Golden Firmware without the risk of bricking the FEB, but it implies a physical access to the board to connect the JTAG programming cable.

To write directly the firmware in the FPGAs, you will need the firmware in the .sof format (directly produced by the firmware compilation).

To write in the EEPROM, you will need the firmware in the .jic format.

File conversion

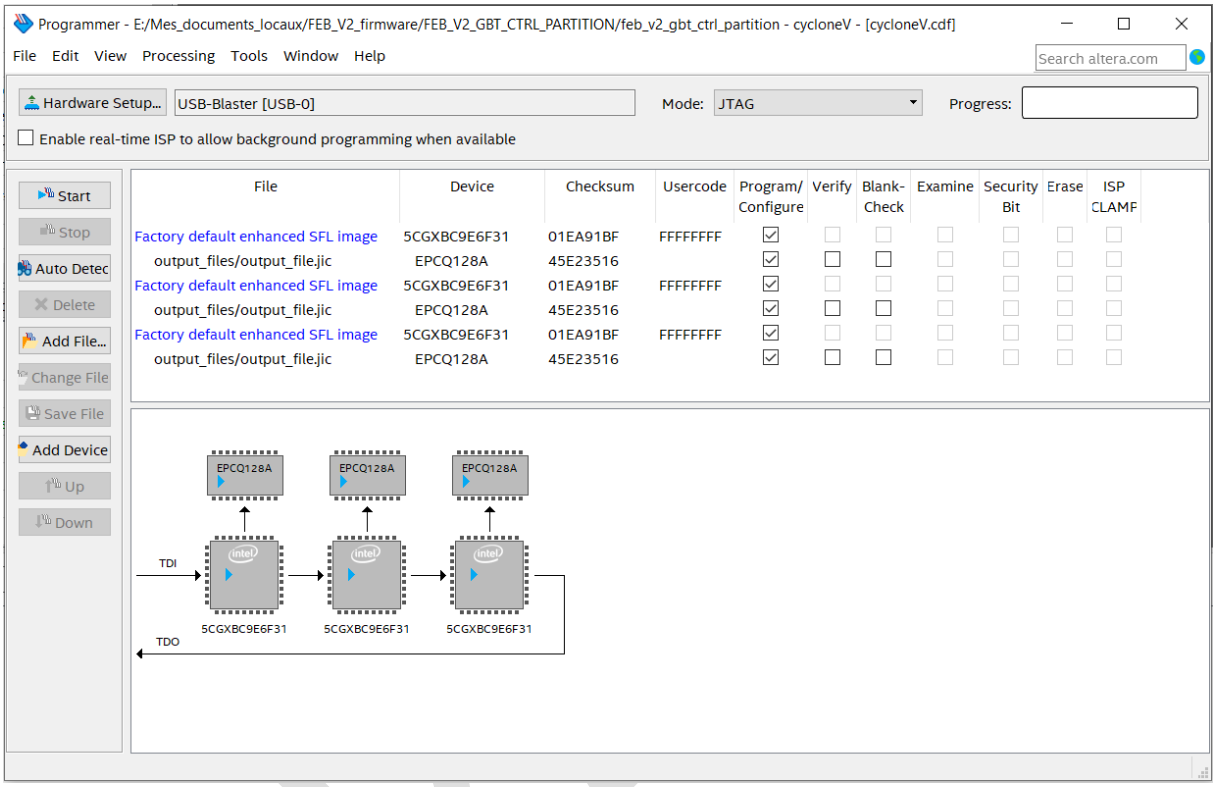
To convert the .sof file in a .jic file, you can use the Quartus converter tool ("File" -> "Convert Programming Files").



Component reconfiguration

Once you have the file you need, you can launch the Quartus programmer tool and recreate the JTAG chain topology (the 3 FPGAs and the 3 attached EEPROM).

To write an EEPROM content, the linked FPGA has to be reconfigured with the factory firmware which includes the Flash loader IP.



Once the new firmware is loaded in the EEPROM, the N_CONFIG sequence (driven by the SCA GPIOs) must be performed again to get the FPGAs configured.

Flash firmware edition through GBT frames (optical link)

This is a fully remote operation but as it can be risky for the flash content integrity it is strongly advised to only edit the Application Firmware part of the flash if you don't have a physical access to the board (if a corruption happens on the Golden Firmware part of the flash, the only way to revive the FEB is to perform a new configuration through the JTAG connector).

For this operation you need to get the .rpd file of the firmware, which is the raw binary content of the flash memory. Using the .map file you can check the start address and the stop address of the Application Firmware you want to load.

First lines of a .map file for a given firmware:

BLOCK	START ADDRESS	END ADDRESS
Page_0	0x00000000	0x0057370D
Page_1	0x00800000	0x00D7370D

…

- All the addresses in this file are byte addresses

…

The flash content is paginated, each page (firmware content) starts at the beginning of each halves of the memory.

- Page_0 starting at address 0 is the Golden Firmware location.
- Page_1 starting at the byte address 0x800000 is the Application Firmware location.

To update the Application Firmware, one needs to:

1. Open the .rpd file in bytemode
2. Write the content of the .rpd files from byte 0x800000 to byte 0xD7370D in the flash from the byte address 0x800000 to the byte address 0xD7370D.

Note that for the Application Firmware the start address is always 0x800000 but the end address can change, this is why it is mandatory to check the range in the .map file.

Alternatively, you can choose to always copy the full second part of the .rpd file (from byte 0x800000 to byte 0xFFFFFF), and in this case you don't need the .map file.

To get more information about how to edit the flash content, refer to the Flash Interface Slow Control slave section.

FPGAs Slow control communication check

The slow control slave 0 can be freely used to test communication with the 3 FPGAs.

The first 16 registers of this slave (addressed from 0x0000 to 0x000F) can be written and read back to test the link behavior (these registers are reserved for test purpose and the values are ignored internally).

The read only register 16 (addressed 0x0010) stores the ID of its FPGA:

- FPGA_LEFT_ID = 0
- FPGA_MIDDLE_ID = 1
- FPGA_RIGHT_ID = 2

FPGA test frame examples:

```
0x0007 0x0000 0x0010 0x0000 0x0000    #read ID of the 3 FPGAs
0x0002 0x0100 0x0000 0xAAAA 0x0000    #write 0xAAAA in the reg0 of FPGA 1 (middle)
0x0002 0x0000 0x0000 0x0000 0x0000    #read reg0 of FPGA 1
```

TDC control

The TDC configuration is made by writing registers in the slave 0x03 of the FPGAs. During initialization (after a FPGA reset) a default Look Up Table is loaded in the TDC, making it ready to perform tests.

You can easily enable the TDC Channel 32 (BC0 loopback) with these 3 frames:

```
0x0002 0x0100 0x0300 0x0001 0x0000 #Enable TDC (reg0 bit0) for FPGA 1
0x0002 0x0100 0x0307 0x0001 0x0000 #Enable TDC_Channel_32 (reg7 bit0) for FPGA 1
0x0002 0x0100 0x0301 0x0001 0x0000 #CMD Valid (reg1 bit0) for FPGA 1
```

When this configuration is done, sending a BC0 through the fast control field of the downlink frame should generate a data frame on the uplink.

```
0x4000 0x0000 0x0000 0x0000 0x0000     #Simple downlink frame to generate BC0
```

It is also possible to send continuously this frame (at the GBT frame rate = 40MHz) to check that the timestamp difference of two consecutive TDC data is 25ns.

Important Note:

When the FPGA process and send data continuously at high rate, it can be impossible to read slow control registers (TDC data and slow control replies are multiplexed on the same uplink, with a higher priority for data). This doesn't affect write transactions as the downlink bandwith is always dedicated to slow control.

The easiest way to regain control over the bandwidth is to disable TDC module:

```
0x0007 0x0100 0x0300 0x0000 0x0000  #Disable TDC (reg0 bit0) for the 3 FPGAs
```

This mutes data generation in the FPGAs.

The TDC readout module implements a data counter which counts generated TDC data for each channel during a configurable time window. The time window parameter is 32bits wide with a LSB of 10ns.

Example frames to use data counter feature:

```
0x0002 0x0101 0x030A 0x4240 0x000F   #Configure a time window of 10ms for FPGA 1
0x0002 0x0100 0x030C 0x0001 0x0000   #Start Data counter (regC bit0) for FPGA 1
0x0002 0x0001 0x0357 0x0000 0x0000   #Read the Channel32 (BC0) data counter for FPGA1
0x0002 0x0043 0x0317 0x0000 0x0000   #Read the 34 channels data counters (68 registers)
```

This kind of test should ensure the TDC channels are generating the correct amount of data.

In addition to the Resync and BC0 loopback features, you can also generate TDC data in an easy way by selecting the injection mode 0b001. This make the input of the 34 TDC channels fed by a 1kHz clock:

```
0x0002 0x0100 0x0308 0x0001 0x0000   #Set the TDC debug injection mode to "001"
```

PETIROC configuration

Once TDC module is operational for data taking, the 2 PETIROCs can be enabled and configured. They are accessible through 2 similar slaves: 0x01 for TOP ASIC and 0x02 for BOTTOM ASIC

Example frames to enable the TOP PETIROC managed by the middle FPGA:

```
0x0002 0x0100 0x0102 0x0010 0x0000   #Disable PETIROC Active low RESET
0x0002 0x0100 0x0104 0x0001 0x0000   #Enable trigger auto-reset state machine
```

The PETIROC needs to be configured before utilization. This configuration is made by writing the parameters in the dedicated 41 FPGA registers and then pushing all the content to the PETIROC thanks to a shift register.

Example frames to load and reset the TOP PETIROC managed by FPGA 1:

```
0x0002 0x0100 0x0101 0x0001 0x0000    #PETIROC configuration reset request frame
0x0002 0x0100 0x0100 0x0001 0x0000    #PETIROC configuration load request frame
```

In our application, the only things that needs to be changed during calibration stage are:

- mask_discri_time_ch (for each channel), acts as a channel enable.
- 6b_dac_ch (for each channel), 6bits DAC value used to compensate channel differences.
- 10b_dac_vth_discri_time (common to every channel), 10bits DAC global channel offset.

FEBv2_r2 detailed boot sequence

WARNING: This section refers to an old revision of the FEB (FEBv2r2)

In the following pages, you can find the detailed transcript of the sequence that must be followed to boot the FEBv2_r2 with the FW v4.3 (currently the last available version). This sequence is tested and optimized to allow the utilization of up to 30 meters long power wires (using a CAEN Easy Crate regulated power supply and proper sense wires). A 2200µF electrolytic capacitor was added on both 2V and 4V power terminals of the FEB.

Some commands are redundant because this sequence can be run independently of the current state of the board: cold FEB (boot), hot FEB (reboot), faulty/undefined state (recovery reboot).

At the end of this sequence, the FEB is ready to receive the slow control values to set the PETIROC, TDC and other datapath settings.


```
---------- GBT-FPGA initialization
11:07:51.242  GBT-FPGA reseting...
11:07:51.243  wait GBT ready loop
11:07:51.744  wait GBT ready loop
11:07:52.245  GBT-FPGA ready [OK]

========================  SCA configuration  ==============================

---------- GBT External Control frame generator initialization (GBT-FPGA side)
11:07:52.246  SCA command CONNECT
11:07:52.247  SCA command RESET
----------  SCA enable GPIO, I2C_0, I2C_1, JTAG, ADC
11:07:52.247  SCA command 02 to channel 00 with data 1C000000
11:07:52.248  SCA command 04 to channel 00 with data 00000000
11:07:52.248  SCA command 06 to channel 00 with data 18000000
---------- SCA I2C_0 configured freq=100kHz, nbyte=2 sclmode=CMOS
11:07:52.249  SCA command 30 to channel 03 with data 88000000
---------- 11:07:52.249  SCA I2C_1 configured freq=100kHz, nbyte=2 sclmode=CMOS
11:07:52.249  SCA command 30 to channel 04 with data 88000000
----------  SCA GPIO direction configuration :
----------  SCA GPIO 0, 1, 2, 6, 7, 8, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 23, 24, 25, 28, 29, 30 -> input
----------  SCA GPIO 3, 4, 5, 9, 10, 21, 22, 26, 27, 31 -> output
11:07:52.250  SCA command 20 to channel 02 with data 8C600638
----------  SCA JTAG configured freq=1000.0 Hz, nbit=46, rxedge=rising, txedge=rising,
bitorder=lsb_first, idletck=high
11:07:52.251  SCA command 90 to channel 13 with data 00004E1F
11:07:52.251  SCA command 80 to channel 13 with data 0000082E
----------  SCA get ID
11:07:52.252  SCA command D1 to channel 14 with data 00000001
11:07:52.252  SCA chip ID is 00BE7A
----------  SCA GPIO 0, 1, 2 configured interrupt on rising edge
----------  SCA GPIO 11, 12, 13, 14, 15, 16, 17 configured interrupt on falling edge
11:07:52.253  SCA command 60 to channel 02 with data 00000001
11:07:52.253  SCA command 30 to channel 02 with data 0003F807
11:07:52.254  SCA command 40 to channel 02 with data 00000007
```

========== FPGA initialization sequence ==========

---------- GPIO 21 set to 1 (FPGA LEFT soft reset)
11:07:52.255 SCA command 11 to channel 02 with data 00000000
11:07:52.256 SCA command 10 to channel 02 with data 00200000
---------- GPIO 3 set to 1 (FPGA LEFT nconfig)
11:07:52.257 SCA command 11 to channel 02 with data 00000000
11:07:52.258 SCA command 10 to channel 02 with data 00200008
---------- Wait 50 ms after any FPGA configuration
---------- GPIO 31 set to 1 (FPGA RIGHT soft reset)
11:07:52.309 SCA command 11 to channel 02 with data 00000000
11:07:52.309 SCA command 10 to channel 02 with data 80200008
---------- GPIO 5 set to 1 (FPGA RIGHT nconfig)
11:07:52.310 SCA command 11 to channel 02 with data 00000000
11:07:52.310 SCA command 10 to channel 02 with data 80200028
---------- Wait 50 ms after any FPGA configuration
---------- GPIO 26 set to 1 (FPGA MIDDLE soft reset)
11:07:52.361 SCA command 11 to channel 02 with data 00000000
11:07:52.362 SCA command 10 to channel 02 with data 84200028
---------- GPIO 4 set to 1 (FPGA MIDDLE nconfig)
11:07:52.363 SCA command 11 to channel 02 with data 00000000
11:07:52.363 SCA command 10 to channel 02 with data 84200038
---------- Wait 50 ms after any FPGA configuration

---------- GPIO 9 set to 1 (Power_supplies FPGA off)
11:07:52.414 SCA command 11 to channel 02 with data 00000000
11:07:52.415 SCA command 10 to channel 02 with data 84200238
---------- Wait 50 ms after turning off any power supply
---------- GPIO 27 set to 0 (Shutdown 4V)
11:07:52.466 SCA command 11 to channel 02 with data 00000000
11:07:52.467 SCA command 10 to channel 02 with data 84200238
---------- Wait 50 ms after turning off any power supply
---------- GPIO 22 set to 0 (Shutdown 2V)
11:07:52.508 SCA command 11 to channel 02 with data 00000000
11:07:52.509 SCA command 10 to channel 02 with data 84200238
---------- Wait 50 ms after turning off any power supply

---------- GPIO 22 set to 1 (Powerup 2V)
11:07:52.549 SCA command 11 to channel 02 with data 00000000
11:07:52.550 SCA command 10 to channel 02 with data 84600238
---------- Wait 500 ms after turning on 2V
---------- GPIO 27 set to 1 (Powerup 4V)
11:07:53.052 SCA command 11 to channel 02 with data 00000000
11:07:53.052 SCA command 10 to channel 02 with data 8C600238
---------- Wait 500 ms after turning on 4V

---------- Make sure FPGA resets are active (redundant), and turn on FPGA power supply
---------- GPIO 21 set to 1 (FPGA LEFT soft reset)
11:07:53.554 SCA command 11 to channel 02 with data 00000000
11:07:53.555 SCA command 10 to channel 02 with data 8C600238
---------- GPIO 26 set to 1 (FPGA MIDDLE soft reset)
11:07:53.556 SCA command 11 to channel 02 with data 00000000

11:07:53.556 SCA command 10 to channel 02 with data 8C600238
---------- GPIO 31 set to 1 (FPGA RIGHT soft reset)
11:07:53.557 SCA command 11 to channel 02 with data 00000000
11:07:53.558 SCA command 10 to channel 02 with data 8C600238

---------- GPIO 9 set to 1 (Power_supplies FPGA ON)
11:07:53.558 SCA command 11 to channel 02 with data 00000000
11:07:53.559 SCA command 10 to channel 02 with data 8C600238

---------- staggered release of FPGA resets

---------- GPIO 21 set to 1 (FPGA LEFT soft reset), redundant
11:07:53.660 SCA command 11 to channel 02 with data 00000000
11:07:53.661 SCA command 10 to channel 02 with data 8C600238
---------- GPIO 21 set to 0
11:07:53.661 SCA command 11 to channel 02 with data 00000000
11:07:53.662 SCA command 10 to channel 02 with data 8C400238
---------- Wait 50 ms after every FPGA reset

---------- GPIO 26 set to 1 (FPGA MIDDLE soft reset), redundant
11:07:53.713 SCA command 11 to channel 02 with data 00000000
11:07:53.714 SCA command 10 to channel 02 with data 8C400238
---------- GPIO 26 set to 0
11:07:53.714 SCA command 11 to channel 02 with data 00000000
11:07:53.715 SCA command 10 to channel 02 with data 88400238
---------- Wait 50 ms after every FPGA reset

---------- GPIO 31 set to 1 (FPGA RIGHT soft reset), redundant
11:07:53.766 SCA command 11 to channel 02 with data 00000000
11:07:53.767 SCA command 10 to channel 02 with data 88400238
---------- GPIO 31 set to 0
11:07:53.767 SCA command 11 to channel 02 with data 00000000
11:07:53.768 SCA command 10 to channel 02 with data 08400238
---------- Wait 50 ms after every FPGA reset

========== Check all sensors sequence ==========

---------- Read GPIO
11:07:53.819 SCA command 01 to channel 02 with data 00000000
---------- Read ADC_RSSI
11:07:53.820 SCA command 50 to channel 14 with data 00000000
11:07:53.820 SCA command 02 to channel 14 with data 00000001
---------- Read ADC_4V_IN
11:07:53.821 SCA command 50 to channel 14 with data 00000015
11:07:53.821 SCA command 02 to channel 14 with data 00000001
---------- Read ADC_4V_IN_CURRENT
11:07:53.833 SCA command 50 to channel 14 with data 0000000D
11:07:53.833 SCA command 02 to channel 14 with data 00000001
---------- Read ADC_2V5_SAFE
11:07:53.834 SCA command 50 to channel 14 with data 00000005
11:07:53.834 SCA command 02 to channel 14 with data 00000001
---------- Read ADC_3V3_VCCIO
11:07:53.846 SCA command 50 to channel 14 with data 00000003
11:07:53.846 SCA command 02 to channel 14 with data 00000001
---------- Read ADC_2V5_VCCIO
11:07:53.847 SCA command 50 to channel 14 with data 00000002
11:07:53.847 SCA command 02 to channel 14 with data 00000001
---------- Read ADC_VH_VCCIO_LEFT
11:07:53.858 SCA command 50 to channel 14 with data 00000011
11:07:53.859 SCA command 02 to channel 14 with data 00000001
---------- Read ADC_VH_VCCIO_MIDDLE
11:07:53.859 SCA command 50 to channel 14 with data 00000012
11:07:53.860 SCA command 02 to channel 14 with data 00000001
---------- Read ADC_VH_VCCIO_RIGHT
11:07:53.860 SCA command 50 to channel 14 with data 00000013
11:07:53.861 SCA command 02 to channel 14 with data 00000001
---------- Read ADC_2V_IN
11:07:53.861 SCA command 50 to channel 14 with data 00000014
11:07:53.862 SCA command 02 to channel 14 with data 00000001
---------- Read ADC_2V_IN_CURRENT
11:07:53.862 SCA command 50 to channel 14 with data 0000000C
11:07:53.863 SCA command 02 to channel 14 with data 00000001
---------- Read ADC_1V5_SAFE
11:07:53.863 SCA command 50 to channel 14 with data 00000004
11:07:53.864 SCA command 02 to channel 14 with data 00000001
---------- Read ADC_1V5_VCCIO
11:07:53.875 SCA command 50 to channel 14 with data 00000001
11:07:53.875 SCA command 02 to channel 14 with data 00000001
---------- Read ADC_1V1_CORE_LEFT
11:07:53.887 SCA command 50 to channel 14 with data 00000006
11:07:53.887 SCA command 02 to channel 14 with data 00000001
---------- Read ADC_1V1_CORE_MIDDLE
11:07:53.888 SCA command 50 to channel 14 with data 00000007
11:07:53.889 SCA command 02 to channel 14 with data 00000001
---------- Read ADC_1V1_CORE_RIGHT
11:07:53.889 SCA command 50 to channel 14 with data 00000008
11:07:53.890 SCA command 02 to channel 14 with data 00000001

---------- Read ADC_1V1_VCCE_LEFT
11:07:53.890 SCA command 50 to channel 14 with data 00000009
11:07:53.891 SCA command 02 to channel 14 with data 00000001
---------- Read ADC_1V1_VCCE_MIDDLE
11:07:53.891 SCA command 50 to channel 14 with data 0000000A
11:07:53.892 SCA command 02 to channel 14 with data 00000001
---------- Read ADC_1V1_VCCE_RIGHT
11:07:53.892 SCA command 50 to channel 14 with data 0000000B
11:07:53.893 SCA command 02 to channel 14 with data 00000001
---------- Read ADC_CURRENT_CORE_LEFT
11:07:53.893 SCA command 50 to channel 14 with data 0000000E
11:07:53.894 SCA command 02 to channel 14 with data 00000001
---------- Read ADC_CURRENT_CORE_MIDDLE
11:07:53.894 SCA command 50 to channel 14 with data 0000000F
11:07:53.895 SCA command 02 to channel 14 with data 00000001
---------- Read ADC_CURRENT_CORE_RIGHT
11:07:53.895 SCA command 50 to channel 14 with data 00000010
11:07:53.896 SCA command 02 to channel 14 with data 00000001
---------- LM75_POWER_4V
11:07:53.896 SCA I2C_1 write single byte=00 to addr=48
11:07:53.896 SCA command 82 to channel 04 with data 48000000
11:07:53.907 SCA command DE to channel 04 with data 48000000
11:07:53.919 SCA command 71 to channel 04 with data 00000000
---------- LM75_POWER_2V
11:07:53.919 SCA I2C_1 write single byte=00 to addr=49
11:07:53.919 SCA command 82 to channel 04 with data 49000000
11:07:53.931 SCA command DE to channel 04 with data 49000000
11:07:53.942 SCA command 71 to channel 04 with data 00000000
---------- LM75_FPGA_LEFT
11:07:53.942 SCA I2C_1 write single byte=00 to addr=4A
11:07:53.942 SCA command 82 to channel 04 with data 4A000000
11:07:53.953 SCA command DE to channel 04 with data 4A000000
11:07:53.965 SCA command 71 to channel 04 with data 00000000
---------- LM75_FPGA_MIDDLE
11:07:53.966 SCA I2C_1 write single byte=00 to addr=4B
11:07:53.966 SCA command 82 to channel 04 with data 4B000000
11:07:53.977 SCA command DE to channel 04 with data 4B000000
11:07:53.988 SCA command 71 to channel 04 with data 00000000
---------- LM75_FPGA_RIGHT
11:07:53.988 SCA I2C_1 write single byte=00 to addr=4C
11:07:53.989 SCA command 82 to channel 04 with data 4C000000
11:07:54.000 SCA command DE to channel 04 with data 4C000000
11:07:54.011 SCA command 71 to channel 04 with data 00000000

================ FPGA go to application firmware sequence ================

---------- LEFT jumps to application firmware
11:07:54.012 LEFT write [128, 0, 3, 1] @2700
11:07:54.012 LEFT write [0, 1, 4, 1] @2700
11:07:54.012 LEFT write [0, 1, 6, 1] @2700
---------- Wait 50 ms after any FPGA configuration

---------- RIGHT jumps to application firmware
11:07:54.063 RIGHT write [128, 0, 3, 1] @2700
11:07:54.063 RIGHT write [0, 1, 4, 1] @2700
11:07:54.063 RIGHT write [0, 1, 6, 1] @2700
---------- Wait 50 ms after any FPGA configuration

---------- MIDDLE jumps to application firmware
11:07:54.114 MIDDLE write [128, 0, 3, 1] @2700
11:07:54.114 MIDDLE write [0, 1, 4, 1] @2700
11:07:54.114 MIDDLE write [0, 1, 6, 1] @2700
---------- Wait 50 ms after any FPGA configuration

---------- GPIO 21 set to 1 (FPGA LEFT soft reset)
11:07:54.165 SCA command 11 to channel 02 with data 00000000
11:07:54.165 SCA command 10 to channel 02 with data 08600238
---------- GPIO 21 set to 0
11:07:54.166 SCA command 11 to channel 02 with data 00000000
11:07:54.167 SCA command 10 to channel 02 with data 08400238
---------- Wait 50 ms after every FPGA reset

---------- GPIO 26 set to 1 (FPGA MIDDLE soft reset)
11:07:54.218 SCA command 11 to channel 02 with data 00000000
11:07:54.218 SCA command 10 to channel 02 with data 0C400238
---------- GPIO 26 set to 0
11:07:54.219 SCA command 11 to channel 02 with data 00000000
11:07:54.220 SCA command 10 to channel 02 with data 08400238
---------- Wait 50 ms after every FPGA reset

---------- GPIO 31 set to 1 (FPGA RIGHT soft reset)
11:07:54.270 SCA command 11 to channel 02 with data 00000000
11:07:54.271 SCA command 10 to channel 02 with data 88400238
---------- GPIO 31 set to 0
11:07:54.272 SCA command 11 to channel 02 with data 00000000
11:07:54.273 SCA command 10 to channel 02 with data 08400238
---------- Wait 50 ms after every FPGA reset

---------- check MIDDLE communication
11:07:54.324 MIDDLE write [13738, 41342, 15136, 20750, 20941, 64428, 54208, 7328, 60890, 37778, 57854, 32818, 42567, 13685, 27151, 48324] @0000
11:07:54.326 MIDDLE read [13738, 41342, 15136, 20750, 20941, 64428, 54208, 7328, 60890, 37778, 57854, 32818, 42567, 13685, 27151, 48324] @0000
---------- check MIDDLE runs application firmware
11:07:54.326 MIDDLE write [4, 0, 1] @2702
11:07:54.326 MIDDLE read [0, 1] @2711

---------- check LEFT communication
11:07:54.326 LEFT write [3207, 54857, 487, 18456, 62389, 53969, 40312, 17975, 43002, 24339, 18922, 58925, 13997, 46836, 57978, 42652] @0000
11:07:54.328 LEFT read [3207, 54857, 487, 18456, 62389, 53969, 40312, 17975, 43002, 24339, 18922, 58925, 13997, 46836, 57978, 42652] @0000
---------- check LEFT runs application firmware
11:07:54.328 LEFT write [4, 0, 1] @2702
11:07:54.328 LEFT read [0, 1] @2711

---------- check RIGHT communication
11:07:54.329 RIGHT write [48470, 18148, 6003, 58381, 43810, 33718, 7184, 37968, 4179, 44195, 29091, 49547, 3426, 60289, 18190, 17882] @0000
11:07:54.330 RIGHT read [48470, 18148, 6003, 58381, 43810, 33718, 7184, 37968, 4179, 44195, 29091, 49547, 3426, 60289, 18190, 17882] @0000
---------- check RIGHT runs application firmware
11:07:54.330 RIGHT write [4, 0, 1] @2702
11:07:54.331 RIGHT read [0, 1] @2711

---------- read firmware versions
11:07:54.331 LEFT read [4] @0011
11:07:54.331 LEFT read [3] @0012
11:07:54.332 LEFT read [0] @0010
11:07:54.332 MIDDLE read [4] @0011
11:07:54.332 MIDDLE read [3] @0012
11:07:54.333 MIDDLE read [1] @0010
11:07:54.333 RIGHT read [4] @0011
11:07:54.333 RIGHT read [3] @0012
11:07:54.334 RIGHT read [2] @0010

11:07:54.334 FEB boot DONE and CHECKED time : 2.09 seconds

FEBv2_r2 Data channel mapping

WARNING: Section refers to an old revision of the FEB (FEBv2r2)

FEBv2r2 Strip/PETIROC/TDC channel mapping

FPGA LEFT (0)

	PETIROC TOP (Direct Strip)			PETIROC BOTTOM (Return Strip)	
Strip n°	PETIROC Channel	FPGA TDC Channel	RStrip n°	PETIROC Channel	FPGA TDC Channel
15	1	0	15	30	31
14	2	1	14	28	30
13	4	2	13	26	29
12	6	3	12	24	28
11	8	4	11	22	27
10	10	5	10	20	26
9	12	6	9	18	25
8	14	7	8	16	24
7	16	8	7	14	23
6	18	9	6	12	22
5	20	10	5	10	21
4	22	11	4	8	20
3	24	12	3	6	19
2	26	13	2	4	18
1	28	14	1	2	17
0	30	15	0	1	16

FPGA MIDDLE (1)

	PETIROC TOP (Direct Strip)			PETIROC BOTTOM (Return Strip)	
Strip n°	PETIROC Channel	FPGA TDC Channel	RStrip n°	PETIROC Channel	FPGA TDC Channel
31	1	0	31	30	31
30	2	1	30	28	30
29	4	2	29	26	29
28	6	3	28	24	28
27	8	4	27	22	27
26	10	5	26	20	26
25	12	6	25	18	25
24	14	7	24	16	24
23	16	8	23	14	23
22	18	9	22	12	22
21	20	10	21	10	21
20	22	11	20	8	20
19	24	12	19	6	19
18	26	13	18	4	18
17	28	14	17	2	17
16	30	15	16	1	16

FPGA RIGHT (2)

	PETIROC TOP (Direct Strip)			PETIROC BOTTOM (Return Strip)	
Strip n°	PETIROC Channel	FPGA TDC Channel	RStrip n°	PETIROC Channel	FPGA TDC Channel
47	1	0	47	30	31
46	2	1	46	28	30
45	4	2	45	26	29
44	6	3	44	24	28
43	8	4	43	22	27
42	10	5	42	20	26
41	12	6	41	18	25
40	14	7	40	16	24
39	16	8	39	14	23
38	18	9	38	12	22
37	20	10	37	10	21
36	22	11	36	8	20
35	24	12	35	6	19
34	26	13	34	4	18
33	28	14	33	2	17
32	30	15	32	1	16

For each FPGA:	TDC Channel 32	BC0
	TDC Channel 33	Resync

PETIROC configuration reference

WARNING: Section refers to an old revision of the FEB (FEBv2r2)

In the following tab, you can find information on parameters to configure the PETIROC slow control registers with a proposed (tested) value. The values that need to be controlled/changed accordingly to the wanted FEBv2_r2 utilization are highlighted in **green**.

Name	Bit index	Size	Order	Proposed value
mask_discri_charge_ch0	0	1		1
mask_discri_charge_ch1	1	1		1
mask_discri_charge_ch2	2	1		1
mask_discri_charge_ch3	3	1		1
mask_discri_charge_ch4	4	1		1
mask_discri_charge_ch5	5	1		1
mask_discri_charge_ch6	6	1		1
mask_discri_charge_ch7	7	1		1
mask_discri_charge_ch8	8	1		1
mask_discri_charge_ch9	9	1		1
mask_discri_charge_ch10	10	1		1
mask_discri_charge_ch11	11	1		1
mask_discri_charge_ch12	12	1		1
mask_discri_charge_ch13	13	1		1
mask_discri_charge_ch14	14	1		1
mask_discri_charge_ch15	15	1		1
mask_discri_charge_ch16	16	1		1
mask_discri_charge_ch17	17	1		1
mask_discri_charge_ch18	18	1		1
mask_discri_charge_ch19	19	1		1
mask_discri_charge_ch20	20	1		1
mask_discri_charge_ch21	21	1		1
mask_discri_charge_ch22	22	1		1
mask_discri_charge_ch23	23	1		1
mask_discri_charge_ch24	24	1		1
mask_discri_charge_ch25	25	1		1
mask_discri_charge_ch26	26	1		1
mask_discri_charge_ch27	27	1		1
mask_discri_charge_ch28	28	1		1
mask_discri_charge_ch29	29	1		1
mask_discri_charge_ch30	30	1		1
mask_discri_charge_ch31	31	1		1
input_dac_ch0	32	8	LSB first	0x80
cmd_input_dac_ch0	40	1		1
input_dac_ch1	41	8	LSB first	0x80
cmd_input_dac_ch1	49	1		1
input_dac_ch2	50	8	LSB first	0x80

cmd_input_dac_ch2	58	1		1
input_dac_ch3	59	8	LSB first	0x80
cmd_input_dac_ch3	67	1		1
input_dac_ch4	68	8	LSB first	0x80
cmd_input_dac_ch4	76	1		1
input_dac_ch5	77	8	LSB first	0x80
cmd_input_dac_ch5	85	1		1
input_dac_ch6	86	8	LSB first	0x80
cmd_input_dac_ch6	94	1		1
input_dac_ch7	95	8	LSB first	0x80
cmd_input_dac_ch7	103	1		1
input_dac_ch8	104	8	LSB first	0x80
cmd_input_dac_ch8	112	1		1
input_dac_ch9	113	8	LSB first	0x80
cmd_input_dac_ch9	121	1		1
input_dac_ch10	122	8	LSB first	0x80
cmd_input_dac_ch10	130	1		1
input_dac_ch11	131	8	LSB first	0x80
cmd_input_dac_ch11	139	1		1
input_dac_ch12	140	8	LSB first	0x80
cmd_input_dac_ch12	148	1		1
input_dac_ch13	149	8	LSB first	0x80
cmd_input_dac_ch13	157	1		1
input_dac_ch14	158	8	LSB first	0x80
cmd_input_dac_ch14	166	1		1
input_dac_ch15	167	8	LSB first	0x80
cmd_input_dac_ch15	175	1		1
input_dac_ch16	176	8	LSB first	0x80
cmd_input_dac_ch16	184	1		1
input_dac_ch17	185	8	LSB first	0x80
cmd_input_dac_ch17	193	1		1
input_dac_ch18	194	8	LSB first	0x80
cmd_input_dac_ch18	202	1		1
input_dac_ch19	203	8	LSB first	0x80
cmd_input_dac_ch19	211	1		1
input_dac_ch20	212	8	LSB first	0x80
cmd_input_dac_ch20	220	1		1
input_dac_ch21	221	8	LSB first	0x80
cmd_input_dac_ch21	229	1		1
input_dac_ch22	230	8	LSB first	0x80
cmd_input_dac_ch22	238	1		1
input_dac_ch23	239	8	LSB first	0x80
cmd_input_dac_ch23	247	1		1
input_dac_ch24	248	8	LSB first	0x80
cmd_input_dac_ch24	256	1		1
input_dac_ch25	257	8	LSB first	0x80

cmd_input_dac_ch25	265	1		1
input_dac_ch26	266	8	LSB first	0x80
cmd_input_dac_ch26	274	1		1
input_dac_ch27	275	8	LSB first	0x80
cmd_input_dac_ch27	283	1		1
input_dac_ch28	284	8	LSB first	0x80
cmd_input_dac_ch28	292	1		1
input_dac_ch29	293	8	LSB first	0x80
cmd_input_dac_ch29	301	1		1
input_dac_ch30	302	8	LSB first	0x80
cmd_input_dac_ch30	310	1		1
input_dac_ch31	311	8	LSB first	0x80
cmd_input_dac_ch31	319	1		1
input_dac_ch_dummy	320	8	LSB first	0x80
mask_discri_time_ch0	328	1		1
mask_discri_time_ch1	329	1		1
mask_discri_time_ch2	330	1		1
mask_discri_time_ch3	331	1		1
mask_discri_time_ch4	332	1		1
mask_discri_time_ch5	333	1		1
mask_discri_time_ch6	334	1		1
mask_discri_time_ch7	335	1		1
mask_discri_time_ch8	336	1		1
mask_discri_time_ch9	337	1		1
mask_discri_time_ch10	338	1		1
mask_discri_time_ch11	339	1		1
mask_discri_time_ch12	340	1		1
mask_discri_time_ch13	341	1		1
mask_discri_time_ch14	342	1		1
mask_discri_time_ch15	343	1		1
mask_discri_time_ch16	344	1		1
mask_discri_time_ch17	345	1		1
mask_discri_time_ch18	346	1		1
mask_discri_time_ch19	347	1		1
mask_discri_time_ch20	348	1		1
mask_discri_time_ch21	349	1		1
mask_discri_time_ch22	350	1		1
mask_discri_time_ch23	351	1		1
mask_discri_time_ch24	352	1		1
mask_discri_time_ch25	353	1		1
mask_discri_time_ch26	354	1		1
mask_discri_time_ch27	355	1		1
mask_discri_time_ch28	356	1		1
mask_discri_time_ch29	357	1		1
mask_discri_time_ch30	358	1		1
mask_discri_time_ch31	359	1		1

6b_dac_ch0	360	6	LSB first	0x01
6b_dac_ch1	366	6	LSB first	0x01
6b_dac_ch2	372	6	LSB first	0x01
6b_dac_ch3	378	6	LSB first	0x01
6b_dac_ch4	384	6	LSB first	0x01
6b_dac_ch5	390	6	LSB first	0x01
6b_dac_ch6	396	6	LSB first	0x01
6b_dac_ch7	402	6	LSB first	0x01
6b_dac_ch8	408	6	LSB first	0x01
6b_dac_ch9	414	6	LSB first	0x01
6b_dac_ch10	420	6	LSB first	0x01
6b_dac_ch11	426	6	LSB first	0x01
6b_dac_ch12	432	6	LSB first	0x01
6b_dac_ch13	438	6	LSB first	0x01
6b_dac_ch14	444	6	LSB first	0x01
6b_dac_ch15	450	6	LSB first	0x01
6b_dac_ch16	456	6	LSB first	0x01
6b_dac_ch17	462	6	LSB first	0x01
6b_dac_ch18	468	6	LSB first	0x01
6b_dac_ch19	474	6	LSB first	0x01
6b_dac_ch20	480	6	LSB first	0x01
6b_dac_ch21	486	6	LSB first	0x01
6b_dac_ch22	492	6	LSB first	0x01
6b_dac_ch23	498	6	LSB first	0x01
6b_dac_ch24	504	6	LSB first	0x01
6b_dac_ch25	510	6	LSB first	0x01
6b_dac_ch26	516	6	LSB first	0x01
6b_dac_ch27	522	6	LSB first	0x01
6b_dac_ch28	528	6	LSB first	0x01
6b_dac_ch29	534	6	LSB first	0x01
6b_dac_ch30	540	6	LSB first	0x01
6b_dac_ch31	546	6	LSB first	0x01
EN_10bits_DAC	552	1		1
PP_10bits_DAC	553	1		1
10b_dac_vth_discri_charge	554	10	MSB first	0x000
10b_dac_vth_discri_time	564	10	MSB first	0x1F4
EN_ADC	574	1		0
PP_ADC	575	1		0
sel_startb_ramp_ADC_ext	576	1		0
usebcompensation	577	1		0
EN_bias_DAC_delay	578	1		1
PP_bias_DAC_delay	579	1		1
EN_bias_ramp_delay	580	1		0
PP_bias_ramp_delay	581	1		0
8b_dac_delay	582	8	LSB first	0x00
EN_discri_delay	590	1		1

PP_discri_delay	591	1		1
PP_temp_sensor	592	1		0
EN_temp_sensor	593	1		0
EN_bias_pa	594	1		1
PP_bias_pa	595	1		1
EN_bias_discri	596	1		1
PP_bias_discri	597	1		1
cmd_polarity	598	1		0
latch_discri	599	1		1
EN_bias_6b_dac	600	1		1
PP_bias_6b_dac	601	1		1
EN_bias_tdc	602	1		0
PP_bias_tdc	603	1		0
ON_OFF_input_dac	604	1		1
EN_bias_charge	605	1		0
PP_bias_charge	606	1		0
cf_100fF	607	1		0
cf_200fF	608	1		0
cf_2_5pF	609	1		0
cf_1_25pF	610	1		0
EN_bias_sca	611	1		0
PP_bias_sca	612	1		0
EN_bias_discri_charge	613	1		0
PP_bias_discri_charge	614	1		0
EN_bias_discri_adc_time	615	1		0
PP_bias_discri_adc_time	616	1		0
EN_bias_discri_adc_charge	617	1		0
PP_bias_discri_adc_charge	618	1		0
DIS_razchn_int	619	1		1
DIS_razchn_ext	620	1		0
SEL_80M	621	1		0
EN_80M	622	1		0
EN_slow_lvds_rec	623	1		1
PP_slow_lvds_rec	624	1		1
EN_fast_lvds_rec	625	1		1
PP_fast_lvds_rec	626	1		0
EN_transmitter	627	1		0
PP_transmitter	628	1		0
ON_OFF_1mA	629	1		1
ON_OFF_2mA	630	1		1
NC1	631	1		0
ON_OFF_ota_mux	632	1		0
ON_OFF_ota_probe	633	1		0
DIS_trig_mux	634	1		1
EN_NOR32_time	635	1		0
EN_NOR32_charge	636	1		0

DIS_triggers	637	1		0	
EN_dout_oc	638	1		0	
EN_transmit	639	1		0	
PA_Ccomp<0>	640	1		0	2B & 2C only
PA_Ccomp<1>	641	1		0	
PA_Ccomp<2>	642	1		0	
PA_Ccomp<3>	643	1		1	
NC2	644	1		1	
NC3	645	1		0	
NC4	646	1		0	
Choice_trigger_out	647	1		0	
Delay_reset_trigger	648	4	LSB first	0x0	2C only
NC5	652	1		0	
NC6	653	1		0	
NC7	654	1		0	
En_reset_trigger_delay	655	1		0	
Delay_reset_ToT	656	4	LSB first	0x0	
NC8	660	1		0	
NC9	661	1		0	
NC10	662	1		0	
EN_reset_ToT_delay	663	1		0	

PETIROC channel auto-reset module (OLD FW ONLY)

WARNING: This feature is removed for FEB FW v4.0 and higher

Only PETIROC 2C with internal auto-reset is supported with the last FW.

PETIROC 2A and PETIROC 2B ASICs do not implement internal automatic reset for their trigger system. Once a signal is detected for one of its channels and the corresponding trigger is transmitted to the FPGA, the channel is not active until a reset sequence is applied on the dedicated pins of the PETIROC. FPGAs implement a configurable state machine in order to automatize this sequence.

This channel reset sequence is common for every channel of a PETIROC ASIC. This introduces two major constraints:

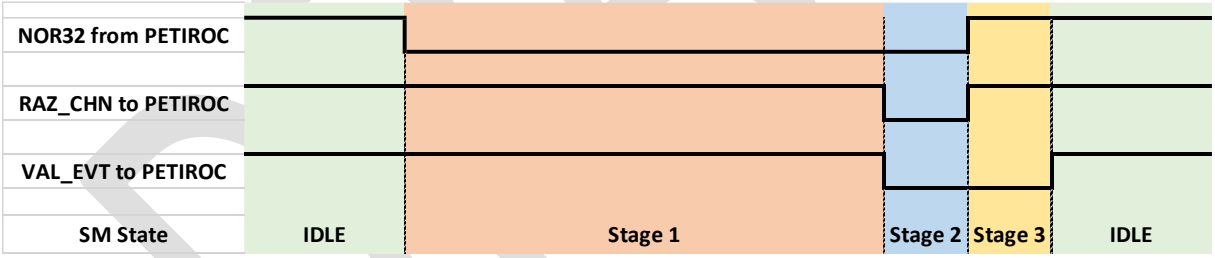
- During the reset sequence every channel is inhibited (even the ones that have not been triggered), causing a dead time for the whole PETIROC.
- Before resetting, system needs to be sure the trigger generated from the other side of the strip has been received by the TDC. It was decided to manage this problem by adding a waiting stage, before the actual reset stages within the state machine.

PETIROC pins

This mechanism relies on three PETIROC pins: the NOR32 output pin, which indicates at least one trigger occurred in the ASIC and the RAZ_CHN and VAL_EVT input pins, which are used to command the channel reset.

- NOR32: When low, this signal indicates that at least one of the PETIROC channels is triggered.
- RAZ_CHN: Active low reset of the PETIROC channel triggers.
- VAL_EVT: When low, the PETIROC channels are masked (disabled) and they can't be triggered even if there is a signal coming from the strips.

State machine sequencing



- IDLE: All channels are untriggered, reset pins are inactive.
- Stage 1: At least one trigger has been sent to the FPGA TDC, this is a waiting state to be sure the signal from the other side of the strip has been captured.
- Stage 2: PETIROC trigger outputs are reset, and every channel of the ASIC are disabled.
- Stage 3: Reset signal is deasserted but the channel masking is still active to prevent problem of retriggering.

Minimal duration for each state:
IDLE: 5ns
Stage 1: 15ns
Stage 2: 5ns
Stage 3: 5ns

It is possible to increase the duration of Stage 1, Stage 2 and Stage 3 by adding 5ns steps by the dedicated slow control registers. Especially, it is recommended to set the total duration of Stage 1 to 30ns (=add 3 steps) in order to be sure that the signal from the opposite side of the strip has been correctly processed before the reset.