



Doc No: CTRL-FOFB-CC-0001
Issue: 1.0
Date: May 2007
Page: 1 of 42



Diamond Light Source Ltd
Rutherford Appleton Laboratory
Chilton, Didcot
Oxfordshire OX11 0QX
United Kingdom
www.diamond.ac.uk

May 2007

Diamond Light Source Fast Orbit Feedback Communication Controller Specification and Design

Written by: I. S. Uzun,
FPGA Development Engineer, DLS

Approved by: M. T. Heron,
Head of Controls Group, DLS

Document Change Record

Issue	Date	Modified Section/Sheet	Comment
0.1	09 Nov 2005	All	First draft.
1.0	17 May 2007	All	Design updates.



1.	Introduction	6
2.	FOFB Communication Overview	7
2.1	System Overview	7
2.2	Wiring Topology	8
2.3	Distribution of BPM Values	9
2.4	Physical Realisation	9
2.4.1	Hardware Components	9
2.4.2	FOFB Communication Network	10
3.	FOFB Communication Controller Design	11
3.1	Communication Controller Design Overview	11
3.2	Packet Format	11
4.	VHDL Implementation	13
4.1.1	BPM Communication Controller Top-Level VHDL Module	14
4.1.2	Libera Core Interface	17
4.1.2.1	Communication Controller Configuration Interface	17
4.1.2.2	Fast Acquisition Data Interface	20
4.1.3	BPM Communication Controller	22
4.1.3.1	Arbite/Multiplexer (ArbMux) Module (fofb_cc_arbmux.vhd)	22
4.1.3.2	Implementation Details	23
4.1.3.3	Forward or Discard (FoD) Module (fofb_cc_fod.vhd)	24
4.1.3.4	Implementation Details	25
4.1.3.5	TX and TX fifos (fofb_cc_rx/tx_fifo.ngc)	27
4.1.4	Rocket IO Interface Module (fofb_cc_mgt_interface.vhd)	28
4.1.4.1	Implementation Details	30
4.2	CC Design on the PMC Modules	35
4.3	Compile time user definitions	35
4.4	X and Y position array read interface	35
5.	Communication Controller Management	36
5.1	Link Status Query and Fault Monitoring	37
5.2	Test Functionality	38
6.	Design Interface and Integration	38
7.	References	42

Figure 1 Libera (BPM) from Instrumentation Technologies.....	6
Figure 2 Storage Ring Cell Structure	7
Figure 3. FOFB communication network topologies. (a) Optimised Ring topology, (b) Torus Mesh topology.	8
Figure 4. FPGA-based PMC-SFP module from MicroResearch.	10
Figure 5 Communication controller block diagram.	14
Figure 6. BPM Communication Controller top-level VHDL module entity declaration.	15
Figure 7. Fast Acquisition Data Transfer Timing.....	21
Figure 8. Fast Acquisition Data Transfer Timing.....	21
Figure 9. ArbMux module VHDL entity declaration.	22
Figure 10. ArbMux state machine.	23
Figure 11. FoD module VHDL entity declaration.....	24
Figure 12. FoD module architectural block diagram.	26
Figure 13. Forward or Discard Module State Machine.....	27
Figure 14. MGT Interface module VHDL entity declaration.....	28
Figure 15. CC Receiver Module.....	31
Figure 16. Payload transmission timing on CC.	31
Figure 17. Payload receiver timing on CC.	33
Figure 18 Position data interface timing.....	36
Figure 19 EPICS user interface for Fast Feedback Configuration on BPM.....	37

Table 1. BPM Packet Payload Structure.....	11
Table 2. Header field structure.....	12
Table 3 Packet framing for transmission.....	12
Table 4. Libera EBPM FAI Configuration Space Address Decoding in Communication Controller.....	17
Table 5 Communication Controller Configuration Registers.....	17
Table 6. CC status, control and event capture address space.....	19
Table 7. Fast Acquisition Data Blocks.....	20

1. INTRODUCTION

Diamond Light Source is the 3rd generation 3 GeV electron synchrotron currently operational in the UK.

The global Fast Orbit Feedback (FOFB) system for Diamond Light Source is currently being implemented to enhance electron orbit stability. The FOFB system will stabilise the electron beam position towards an ideal electron beam orbit.

The FOFB system consists of 168 electronic Beam Position Monitors (BPMs) with digital signal processing capability, 24 VME-based processors running the feedback algorithm and a FOFB communication network to distribute the beam position values to the feedback processors.

Instrumentation Technologies is delivering the beam position monitor hardware, called Libera (Figure 1), which has a powerful Field Programming Gate Array (FPGA) with fast serial links (RocketIO transceiver). Libera will acquire electron beam position data and provide down-sampled and filtered beam position values to the fast feedback Communication Controller (CC) subsystem at fast feedback rate around 10kHz.

The CC subsystem is central to the data distribution over FOFB communication network. This document defines technical specifications and requirements for the design and implementation of the CC subsystem.



Figure 1 Libera (BPM) from Instrumentation Technologies

2. FOFB COMMUNICATION OVERVIEW

2.1 System Overview

Diamond storage ring consists of 24 identical lattice cells. Each cell has control and instrumentation for that area in an air conditioned, temperature stabilised room, called a CIA. Each CIA houses the racks and instrumentation including a diagnostics rack and a power supply rack.

The diagnostics rack contains 7 Libera BPMs. Each BPM is connected to the set of four button-type pickup electrodes mounted to the vessel. The analog signals picked up by the electrodes are a function of the beam position and intensity. The beam position values are obtained by BPMs from a suitable scaled difference-over-sum calculation. An arm-based single board computer provides EPICS interface for the eBPM.

A BPM also provides 8 SFP connectors which are directly connected to the serial RocketIO transceivers of the Virtex-II Pro FPGA device. These SFP connectors are used to implement the FOFB communication network topologies with serial data rates up to 2.12 Gb/s.

The power supply rack holds an EPICS IOC and a second MVME5500 processor dedicated to feedback algorithm calculation. An FPGA-based PMC-SFP interface module located on the PCI bus of the feedback processor is connected to the FOFB communication network in order to receive beam position values for feedback algorithm calculation.

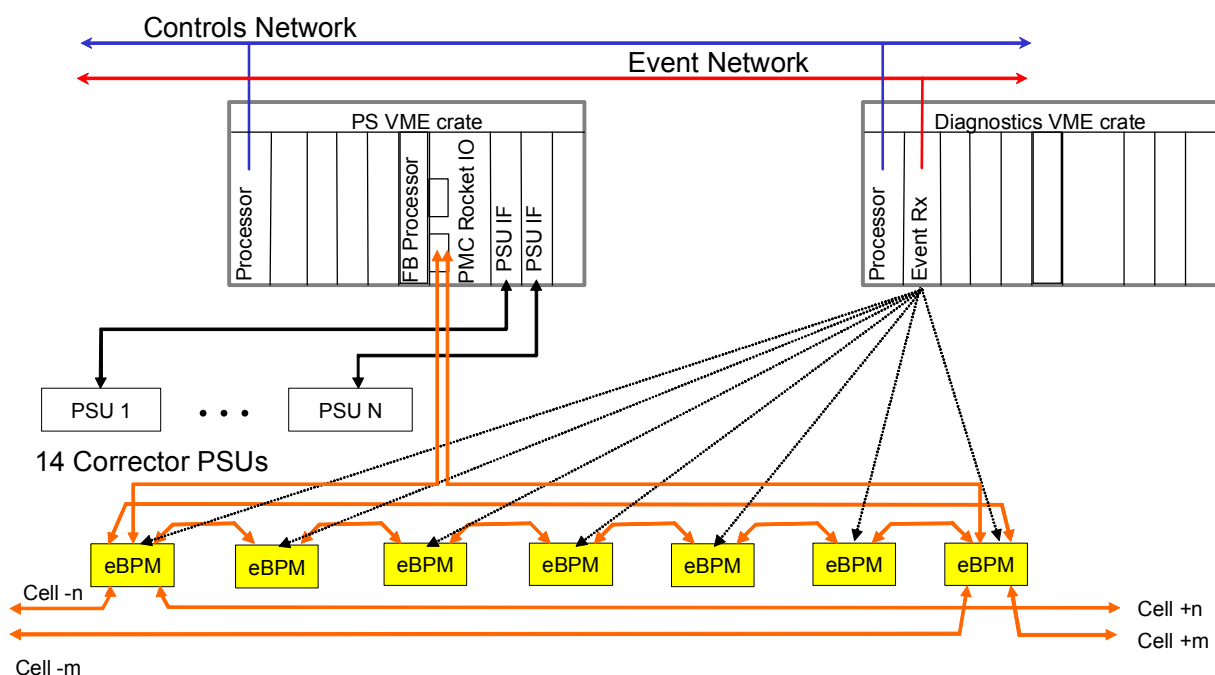


Figure 2 Storage Ring Cell Structure

2.2 Wiring Topology

The CC design is unaware of the network topology so enables a number of options for the physical structure and further support diverse routes so enabling operation to continue with single and multi-link faults. We have considered implementing two network topologies proposed by SuperComputing Systems [1] as shown in Figure 3.

The ring topology resulted from a mapping of the cells and BPMs to the storage ring shape. A simple cell interconnection architecture with bidirectional links will need 3 pairs of fibre from each cell to the central computer room. In figure 3.a the colored links are implementing the cell interconnection at the CSCR patch panel. The BPMs in a cell are connected in a ring including both primary BPMs. The PMC is connected to both primary BPMs.

The ring architecture may be further improved. A fibre count reduction could result if each cell is connected to two unidirectional rings and the rings are superimposed. This basically results in a two dimensional structure. A typical two dimensional mesh is the torus mesh. The torus mesh topology is found in parallel computing environment. It is an interconnect network with loops in two dimensions. The following graphics shows the two primary BPM nodes of a cell and the unidirectional interconnect links via the central computer room. For redundancy reasons, each of the two ring dimension is connected to another primary BPM node. Figure 3.b displays a torus mesh with 4 times 6 cells. Each cell has two primary BPMs, which are connected to either a horizontal or a vertical ring.

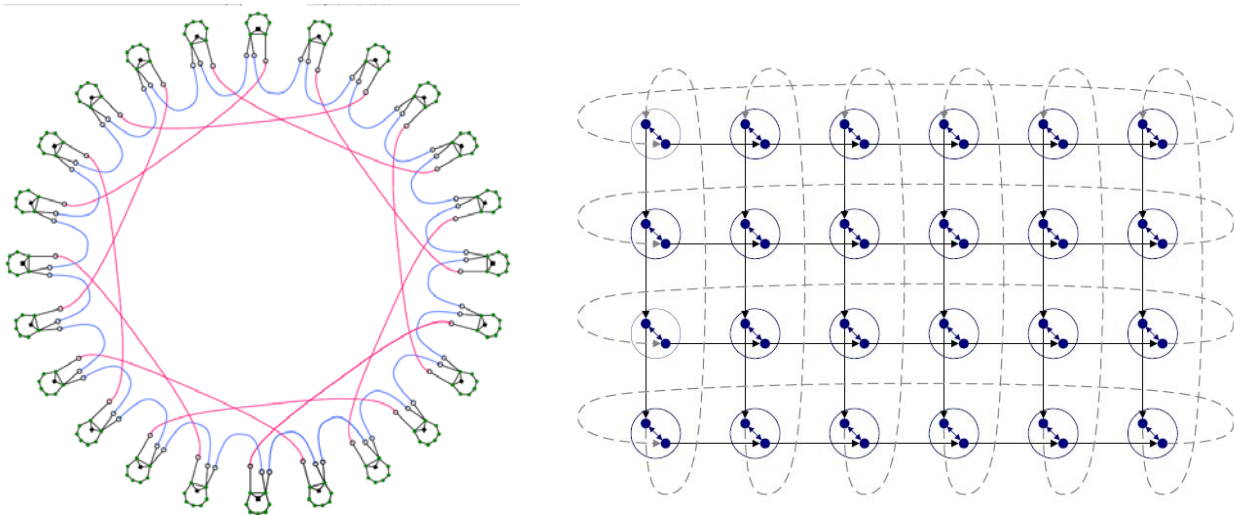


Figure 3. FOFB communication network topologies. (a) Optimised Ring topology, (b) Torus Mesh topology.

2.3 Distribution of BPM Values

The feedback processor in each CIA running the feedback algorithm uses the position values from all 168 BPMs around the storage ring. Therefore, every BPM position value has to be distributed to all 24 feedback processors.

In the current communication scheme, time is divided into time frames of equal length. In the beginning of each time frame, each BPM injects its own position values to the FOFB network, and then starts a forwarding or discarding process of the BPM values that it receives. This is such that a BPM only forwards a packet it receives once. Each BPM position value has to propagate to all 168 BPMs and 24 PMCs on the network before the end of the time frame.

The BPM values can be updated on the feedback processors with the rate of 4 - 20 kHz. This results in the time frame duration of 50 – 250 μ sec. The Libera core will provide a periodic time frame start pulse (along with BPM position values) to the CC subsystem at the rate of 10 kHz.

2.4 Physical Realisation

2.4.1 Hardware Components

Following hardware components have been used in the realisation of the FOFB system:

Libera BPM: Instrumentation Technologies is delivering the beam position monitor hardware, called Libera [2], which has a Xilinx XCV2VP30-5 Virtex-II Pro Field Programming Gate Array (FPGA). Libera will acquire electron beam position data and provide down-sampled and filtered beam position values to the fast feedback Communication Controller (CC) subsystem at fast feedback rate around 10kHz. It also provides eight Small Form factor Pluggable (SFP) slots physically connected to FPGA RocketIO links which are used for the implementation of the FOFB communication network.

Feedback Processors: A second Motorola MVME5500 VME processor with VxWork real-time operating system will be used for running the feedback algorithm.

PMC-SFP Interface Module: The PMC-SFP Interface Module is responsible for distributing the beam position values to the feedback processors. It will be placed on the PMC locations of the feedback processors. The module has a Xilinx XCV2VP30-6 Virtex II Pro FPGA with four RocketIO links used for connecting the module to the FOFB communication network. The module is mainly based on the PMC Event Receiver module from Micro Research [3].

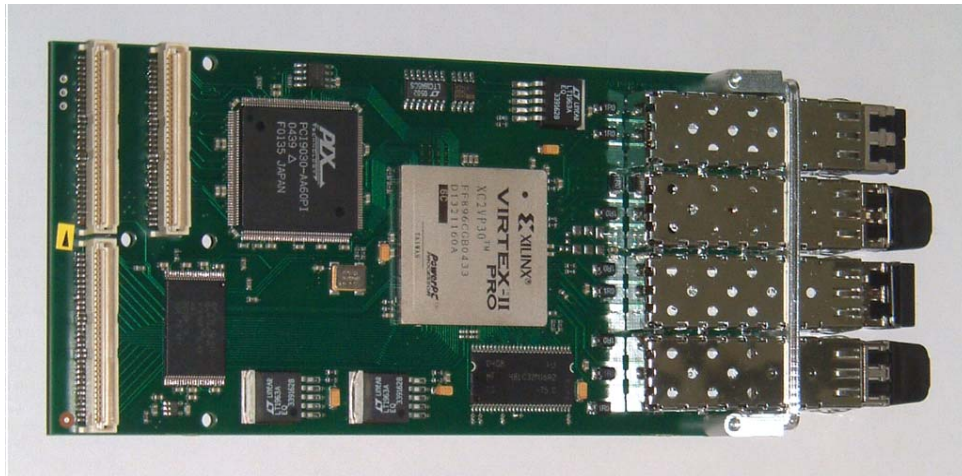


Figure 4. FPGA-based PMC-SFP module from MicroResearch.

2.4.2 FOFB Communication Network

The FOFB communication network is implemented by fast, serial RocketIO links. The BPM digital board provides 106MHz clock source to FPGA to drive the RocketIOs. Therefore, a serial data communication rate of 2.12Gbps ($20 \times 106.25\text{M}$) is achieved. The Rocket IO (RIO) transmit and receive logic is implemented within the FPGA for eight ports. Only the drivers (transceivers) are external and are plugged into SFP (MSA compliant) cages. Transceivers are typically made for fibre optics or copper connections.

The links between BPMs in the diagnostic rack in each cell have been implemented using electrical high-speed copper SFP-SFP/.5m patch cables from CS Electronics[4]. Network topology implemented between BPMs in each cell is shown in Figure 1. “BPM-0” and “BPM-6” are assigned as *Primary BPMs* to be used for connections to PMC-SFP module and Control Systems Computer Room (CSCR) patch panel.

The links between the primary BPMs and PMC-SFP module in the same cell have a length of 5m. Therefore, optical links are required. Agilent Technologies multimode fiber-optic LC transceivers and Duplex LC-LC 5m OM3 fiber patch cables are used to implement these links.

Primary BPMs in each cell are connected to the patch panel in CSCR where the connections between the cells are implemented. These links have total lengths up to 700m, therefore they are implemented using single mode optical transceivers and Duplex LC-LC OS1 fibers.

The connections between cells will be implemented on the patch panel in CSCR. Simplex LC-LC OS1 fibers of length 1m will be used to implement unidirectional cell interconnects.

3. FOFB COMMUNICATION CONTROLLER DESIGN

3.1 Communication Controller Design Overview

In the current communication scheme, time is divided into time frames of equal length of 99.28 μ sec (1/10072). A periodic *time_frame_start* signal is generated by the CC using incoming fast acquisition timing signals from Libera fast acquisition interface. This signal has a period of 1 clock cycle and it indicates the start of a time frame.

The CC design has two processing modules which are Arbiter/Multiplexer (ArbMux) module and the Forward or Discard (FoD) module.

All of the input channels are fed in to ArbMux module. The ArbMux module implements Round-Robin algorithm for input arbitration. A time slot of (# of input channels x PacketSize) clock cycles are allocated for each channel. If the channel has no packet to be processed, it checks other channels. The output packet is sent to FoD module.

The FoD module is responsible for the injection of BPM's own payload in the beginning of each time frame and then processing of received BPM payloads. FoD injects a packet containing its own position values onto the communication network upon the reception of *time_frame_start* pulse. The packet is broadcast to all connected nodes. Following BPM's own packet injection, the FoD starts processing incoming packets from other BPMs on the network. The FoD receives one BPM payload at a time from ArbMux output. The control state machine checks the ForwardedBitArray look-up table to determine if the arbitrated BPM value has already been forwarded in this time frame. If not, it forwards the value by writing the payload on all output TX fifos and sets the corresponding bit in the ForwardedBitArray look-up table. Else it simply discards the BPM payload in order to prevent flooding of the FOFB communication network.

3.2 Packet Format

The 16-byte payload injected to the network by each BPM consists of 4 fields as shown in Table 1.

Table 1. BPM Packet Payload Structure.

Field	Size	Description
Header	32 Bit	BPM ID, status, flags
x-Position	32 Bit	x-Position of the beam [nm]
y-Position	32 Bit	y-Position of the beam [nm]
Time Stamp	32 Bit	Reserved

The Header field has the following structure:

Table 2. Header field structure.

Field	Size	Description
BPM ID	10 Bit	BPM Identification number [0 ..1023]
Source type	3 Bit	i.e: BPM, PMC
Time Frame Start	1 Bit	Signals the start of Time Frame
Time Frame Count	8 Bit	The time frame count value of the BPM. Used for sync. of BPMs
Reserved	10 Bit	Reserved for extended functionality.

The Time Stamp field in the packet payload identifies the number of time frame which the BPM positions values belong to. This value is checked when a packet is received by the communication controller. If the packet does not belong to the current time frame, it is discarded without being written to input queue.

The packets are protected by 32-bit invariant CRC. The RocketIO transmitter computes 4-byte CRC on the payload data. The payload and CRC data is encapsulated by /SOP/ and /EOP/ delimiters before transmission. Table 3 shows the packet framing on the FOFB network links.

Table 3 Packet framing for transmission

Field	Size	Description
/SOP/	2 Bytes	Start Of Packet control symbol
Payload	20 Bytes	BPM Payload
CRC32	4 Bytes	32-bit invariant CRC
/EOF/	2 Bytes	End Of Packet control symbol

If a CRC error occurs, the packet is discarded and no retransmission is issued. The redundancy in the network topology will deliver the same BPM packet through another path. If no packet transmission is in progress, IDLE characters are sent over the link.

4. VHDL IMPLEMENTATION

An overview of the internal structure of the VHDL code is given below where it has been divided in to functional blocks.

- Top-level Design
 - fofb_cc_top.vhd
- Libera Core Interface :
 - fofb_cc_config_intf.vhd
 - fofb_cc_fa_intf.vhd
 - fofb_cc_fa_limiter.vhd
 - fofb_cc_frame_cntrl.vhd
- BPM Communication Controller
 - fofb_cc_arbmux.vhd
 - fofb_cc_fod.vhd
 - fofb_cc_rx_fifo.ngc
 - fofb_cc_tx_fifo.ngc
- RocketIO Transceiver Interface
 - fofb_cc_mgt_interface.vhd
 - phase_align.vhd
 - standard_cc_module.vhd
- Control and Error Monitor
 - fofb_cc_monitor
 - fofb_cc_fifo_rst

4.1.1 BPM Communication Controller Top-Level VHDL Module

fofb_cc_fod.vhd is the top-level VHDL design file where all subsystems are instantiated. A block diagram of top-level CC design is shown in Figure 5. The naming convention of the interface signals has been decided as shown in Figure 8 with Instrumentation Technologies for easier integration.

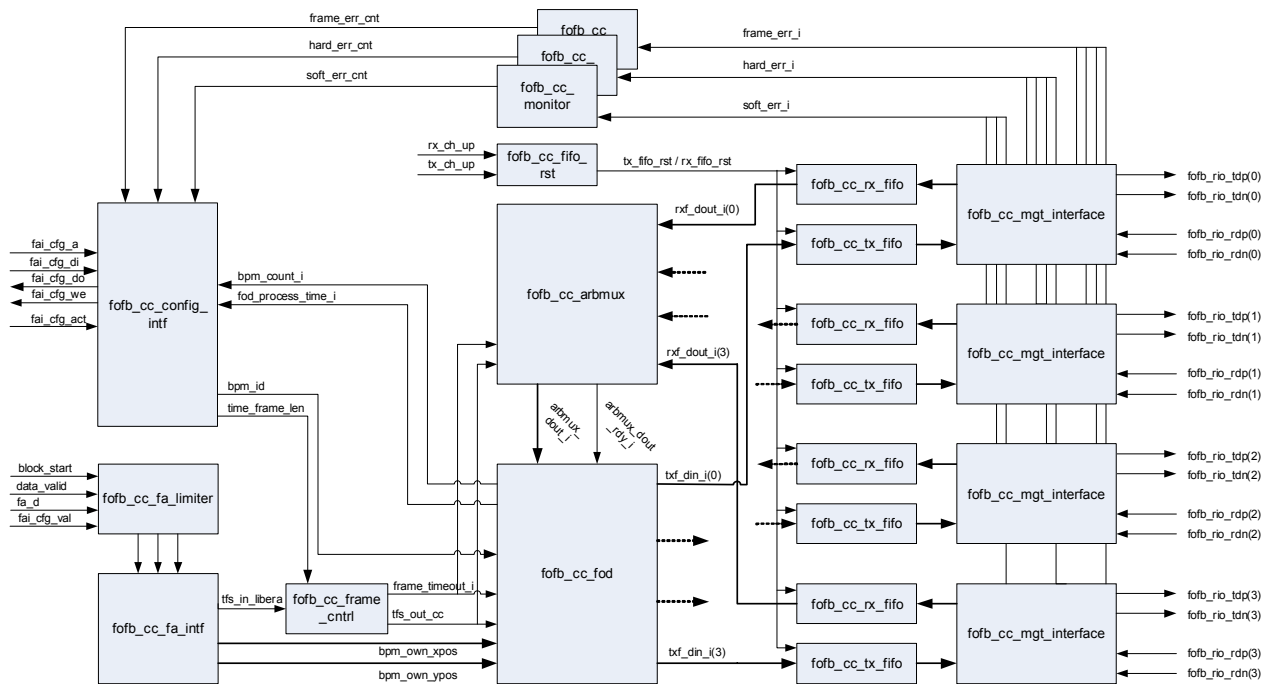


Figure 5 Communication controller block diagram.

```

entity fofb_cc_top is
  port (
    -- clock and reset interface
    adc_clk      : in std_logic;           -- adc rate clock
    mgt_clk      : in std_logic;           -- mgt rate clock
    user_clk_i   : in std_logic;           -- user clock
    adc_rst      : in std_logic;           -- adc clock domain reset
    mgt_rst      : in std_logic;           -- mgt domain reset
    sys_clk      : in std_logic;           -- system clock domain

    -- sbc interface
    sbc_clk      : in std_logic;           -- sbc rate clock
    sbc_rst      : in std_logic;           -- sbc reset
    sbc_cs       : in std_logic;           -- chip select
    sbc_wstb     : in std_logic;           -- write strobe
    sbc_rstb     : in std_logic;           -- read strobe
    sbc_addr     : in std_logic_vector(11 downto 0); -- address
    sbc_dat_i    : in std_logic_vector(31 downto 0); -- input data
    sbc_dat_o    : out std_logic_vector(31 downto 0); -- output data
    sbc_irq_o    : out std_logic;          -- irq to sbc

    -- fast acquisition data interface
    fai_fa_block_start : in std_logic;     -- transfer of block data in progress
    fai_fa_data_valid  : in std_logic;     -- clock signal for incoming data
    fai_fa_d          : in std_logic_vector(15 downto 0); -- fast acquisition data

    -- FOFB communication controller configuration interface
    fai_cfg_a         : out std_logic_vector(10 downto 0); -- address bus for the cfg space
    fai_cfg_do        : out std_logic_vector(15 downto 0); -- data to be written to CFG BRAM
    fai_cfg_di        : in std_logic_vector(31 downto 0); -- data being read from CFG BRAM
    fai_cfg_we        : out std_logic;       -- write Enable signal to BRAM to write Config Data
    fai_cfg_clk       : out std_logic;       -- clock signal from the CC side to CFG BRAM
    fai_cfg_val       : in std_logic_vector(31 downto 0);

    -- serial I/Os for eight RocketIOs on the Libera
    fai_rio_rdp      : in std_logic_vector(RioCount-1 downto 0); -- receive data
    fai_rio_rdn      : in std_logic_vector(RioCount-1 downto 0); -- inverse receive data
    fai_rio_tdp      : out std_logic_vector(RioCount-1 downto 0); -- transmit data
    fai_rio_tdn      : out std_logic_vector(RioCount-1 downto 0); -- inverse transmit data
    fai_rio_tdis     : out std_logic_vector(RioCount-1 downto 0); -- disable transmitters (0:Enable/1:Disable)

    -- Libera trigger interface
    ext_trig_in      : in std_logic;

    -- inverse response matrix coefficient buffer i/o
    coeff_x_addr     : in std_logic_vector(7 downto 0);
    coeff_x_out      : out std_logic_vector(31 downto 0);
    coeff_y_addr     : in std_logic_vector(7 downto 0);
    coeff_y_out      : out std_logic_vector(31 downto 0);

    -- PMC-SFP module interface
    pos_addr         : in std_logic_vector(9 downto 0);           -- position BRAM addr
    pos_dout         : out std_logic_vector(31 downto 0);         -- position data
    frame_timeout    : out std_logic;                             -- timeFrame timeout signal

    -- fofb system heartbeat and event (go, stop) signals
    fofb_heart_beat  : in std_logic;                             -- FOFB heart beat
    fofb_event       : in std_logic_vector(15 downto 0);         -- FOFB Go flag

    -- status info
    fofb_process_time : out std_logic_vector(15 downto 0);
    fofb_bpm_count    : out std_logic_vector(7 downto 0)
  );
end fofb_cc_top;

```

Figure 6. BPM Communication Controller top-level VHDL module entity declaration.



Communication Controller Module Interface

Clock and Reset Inputs

adc_clk	In	1	adc domain clock
mgt_clk	In	1	clock buffer output, used for RocketIO reference clock
user_clk_i	In	1	mgt domain clk used for CC design
adc_rst	In	1	adc domain reset (active high)
mgt_rst	In	1	mgt domain reset (active high)
sys_clk	In	1	sys clock domain for reading stored position data

SBC interface

sbc_clk			For details of SBC interface refer to Instrumentational Technologies's specifications document.
sbc_rst			
sbc_cs			
sbc_wstb			
sbc_rstb			
sbc_adr			
sbc_dat_i			
sbc_dat_o			
sbc_irq_o			

Fast acquisition interface

fai_fa_block_start			For details of SBC interface refer to Instrumentational Technologies's specifications document.
fai_fa_data_valid			
fai_fa_d			

Communication controller configuration BRAM interface

fai_cfg_a			For details of SBC interface refer to Instrumentational Technologies's specifications document.
fai_cfg_do			
fai_cfg_di			
fai_cfg_we			
fai_cfg_clk			
fai_cfg_val			

RocketIO interface

fai_rio_rdp	In	1	RocketIO receive(+) pin
fai_rio_rdn	In	1	RocketIO receive(-) pin
fai_rio_tdp	Out	1	RocketIO transmit (+) pin
fai_rio_tdn	Out	1	RocketIO transmit (-) pin
fai_rio_tdis	Out	8	Enable/Disable RocketIO transceivers.

External trigger

ext_trig_in	In	1	Libera external trigger (used for FA data capture)
-------------	----	---	--

Response matrix data interface

coeff_x_addr	In	8	RM x coefficient address
coeff_x_out	Out	32	RM x coefficient value
coeff_y_addr	In	8	RM y coefficient address
coeff_y_out	Out	32	RM y coefficient value

Position data interface

frame_timeout	Out	1	Frame timeout pulse
pos_addr	In	11	Position data address
pos_dout	Out	32	Position data value

Pmc system status interface

fofb_heart_beat	Out	1	Heart beat pulse at FA rate
fofb_event	Out	32	32-bit event distributed over network
fod_process_time	Out	16	FoD process time in clock cycles
bpm_count	Out	8	Number of nodes on the comm.. network

4.1.2 Libera Core Interface

4.1.2.1 Communication Controller Configuration Interface

fofb_cc_config_intf.vhd module implements configuration interface for the communication controller operation. In order to provide data communication between SBC and CC, a 1K x 32-bit Block RAM (BRAM) at predefined address space has been defined. The address space is divided into four (4) subspaces for configuration and debugging purposes as shown in the table below.

Table 4. Libera EBPM FAI Configuration Space Address Decoding in Communication Controller.

Address range on SBC	Address range on FPGA	Name	Description
0x14028000 – 0x140283FF	0 -255	Communication Controller Configuration	Used for configuring Communication Controller (BPM_ID, FRAME_LENGTH, GOLDEN_ORBIT_X etc)
0x14028400 – 0x140287FF	256-511	Response matrix x elements	During configuration read, this address space is transferred to a separate 256x32-bit BRAM for parallel operation. Use <i>coeff_x_addr</i> and <i>coeff_x_out</i> interface to access this individual BRAM.
0x14028800 – 0x14028BFF	512-767	Response matrix y (z) elements	During configuration read, this address space is transferred to a separate 256x32-bit BRAM for parallel operation. Use <i>coeff_y_addr</i> and <i>coeff_y_out</i> interface to access this individual BRAM.
0x14028C00 – 0x14028FFF	768-1023	Communication Controller Status	This address space is written by the Communication Controller during operation in order to give status information.

4.1.2.1.1 CC Configuration

Following parameters are used for configuration of CC operation.

Table 5 Communication Controller Configuration Registers.

Address range on SBC	Register Name	Set value
0x14028000	bpm_id	BPM ID value. From 0 to 255.
0x14028004	time_frame_length	Time frame length for data distribution in clock cycles in mgt clock domain. Set to <i>required_frame_lenth</i> * 106.25e6. Default value 9000, which gives appx. 90usec. (See Figure 2)
0x14028008	mgt_powerdown	Set individual bit (3:0) to 1 in order to power down RocketIO.
0x1402800C	mgt_loopback	Set individual bits (7:0) to required value in order to loopback RocketIO.
0x14028010	buf_clr_dly	Buffer clear delay for data distribution in clock cycles in mgt clock domain. Set to same value as <i>time_frame_length</i> .
0x14028014	golden_orb_x	Golden orbit x in [nm].
0x14028018	golden_orb_y	Golden orbit y in [nm].



BPM ID : Each BPM has a unique 8-bit BPM ID.

TimeFrameLength : This constant defines the timeout period (in number of clock cycles) for “forward or discard” operation. The actual timeout period can be calculated by $(2/106.25\text{MHz}) * \text{TimeFrameCountDown}$

PowerDown : Each bit of this 4-bit register drives the POWERDOWN input to the four RocketIO transceivers instantiated in the design. POWERDOWN is a single-bit primitive port that allows shutting off the RocketIO transceiver in case it is not needed for the design, or will not be transmitting or receiving for a long period of time.

Bit 3	Bit 2	Bit 1	Bit 0
MGT 4	MGT 3	MGT 2	MGT 1

Loopback : Each 2-bit pair of this 8 bit register is mapped to LOOPBACK ports of the four MGTs in the design. To facilitate testing without the requirement to apply patterns or measure data at gigahertz rates, two programmable loopback features are available in MGTs. One option, serial loopback, places the gigabit transceiver into a state where transmit data is directly fed back to the receiver. An important point to note is that the feedback path is at the output pads of the transmitter. This tests the entirety of the transmitter and receiver.

Bits 6-7	Bits 4-5	Bits 2-3	Bits 0-1
MGT 4	MGT 3	MGT 2	MGT 1

Ports	Mode
00	Normal operation (default value)
01	Serial Loopback
10	Parallel Loopback

4.1.2.1.2 fai_cfg_val Register

A 32-bit *fai_cfg_val* register is used for handshaking between Libera FA module and CC.

fai_cfg_val Register		
0x1402A000	fai_cfg_val	FAI configuration handshake register Bit 0 : Acknowledge configuration data read Bit 1: Position data select (0: real position data, 1: time frame counter) Bit 2 : Unused Bit 3 : Communication controller Enable/Disable Bit 4-31: Time frame count limit value

4.1.2.1.3 Status Registers

The status registers shown in Table 6 allow user applications to monitor the operation of communication controller.

Table 6. CC status, control and event capture address space.

Name	Address	Number of Bits
FPGA Version	0x300	8
System Status	0x301	16
LinkPartner1	0x302	8
LinkPartner2	0x303	8
LinkPartner3	0x304	8
LinkPartner4	0x305	8
LinkUp	0x306	4
TimeFrameCount	0x307	16
SoftErrorCnt1	0x308	16
SoftErrorCnt2	0x309	16
SoftErrorCnt3	0x30A	16
SoftErrorCnt4	0x30B	16
FrameErrorCnt1	0x30C	16
FrameErrorCnt2	0x30D	16
FrameErrorCnt3	0x30E	16
FrameErrorCnt4	0x30F	16
HardErrorCnt1	0x310	16
HardErrorCnt2	0x311	16
HardErrorCnt3	0x312	16
HardErrorCnt4	0x313	16
FodProcessTime	0x314	16
BpmCount	0x315	16

Link Up : This 4-bit register is mapped to four RocketIO transceivers instantiated in the design. Each bit is asserted when channel initialization is complete on the related RocketIO transceivers and link is ready to send data.

LinkPartnerX : This 8-bit register holds the BPM ID of the channel partner which RocketIO transceiver is connected. The content of the register is updated at the half of FOFB operation frequency. If the channel is broken and BPM ID of the partner can not be acquired, 0xFF is written in the register.

SoftErrorCntX : Equipment problems and channel noise can cause errors during RocketIO transceiver channel operation. 8B/10B encoding allows the RocketIO transceiver core to detect all single bit errors and most multi-bit errors that occur in the channel. The *fofb_cc_mgt_interface.vhd* module reports these errors by asserting the SOFT_ERROR signal on every cycle they are detected. This 16-bit counter register holds the number of

SOFT_ERROR pulses for the related RocketIO transceiver channel since the start of Fast Feedback operation.

FrameErrorCntX : *fofb_cc_mgt_interface.vhd* module can also detect errors in frames. Errors of this type include corrupted frames. When the core detects a frame problem, it asserts the FRAME_ERROR signal. This 16-bit counter register holds the number of SOFT_ERROR pulses for the related RocketIO transceiver channel since the start of Fast Feedback operation.

HardErrorCntX : Each RocketIO transceivers are also monitored for hardware errors such as buffer overflow and loss of lock. The *fofb_cc_mgt_interface.vhd* module reports hardware errors by asserting the HARD_ERROR signal. Catastrophic hardware errors can also manifest themselves as burst of soft errors. This 16-bit counter register holds the number of HARD_ERROR pulses for the related RocketIO transceiver channel since the start of Fast Feedback operation.

FodProcessTime : The process time in terms of clock cycles in each time frame is held in this register.

BpmCount : Holds the number of nodes on the network. It is updated on every time frame.

4.1.2.2 Fast Acquisition Data Interface

The *fofb_cc_fa_intf.vhd* module handles fast acquisition data transfer from Libera core module. The parallel data is clocked out from Libera core serially as 32 blocks of 16-bit data. The serially as 32 blocks of 16-bit data as shown in Table 9.

Table 7. Fast Acquisition Data Blocks.

Block No	Meaning
1-8	$I_a, Q_a, I_b, Q_b, I_c, Q_c, I_d, Q_d$
9-12	V_a, V_b, V_c, V_d
13	Sum
14	Q
15	X
16	Y

Fast Acquisition Data Transfer timing is shown in Figure 7. The *BlockStart* signal is asserted at the FA rate which is around 10kHz, and FA data blocks are transferred at the f_{ADC} rate during BlockStart signal is asserted.

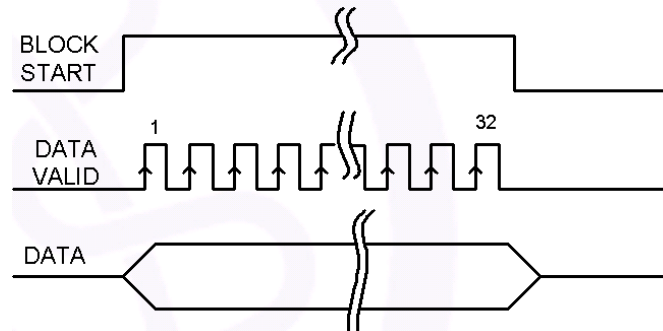


Figure 7. Fast Acquisition Data Transfer Timing.

A 32x16-bit, 1-2 asymmetrical fifo is used in order to manage clock domain crossing and 16-bit to 32-bit conversion. The incoming 16-bit data is written to the fifo at the f_{ADC} rate, while output data is read at CC operation clock rate (106.25Mhz) as blocks of 32-bit.

The *fa_intf_control* state machine manages the write and read interfacing to the fifo. When all fast acquisition data is received and stored in the fifo, read logic starts reading from fifo. The 15th and 16th 32-bit data blocks read correspond to x and y position values, respectively. When the all values are read from the fifo, the *TimeFrameStart* signal is asserted for one clock cycle in order to indicate start of the time frame (Figure 8).

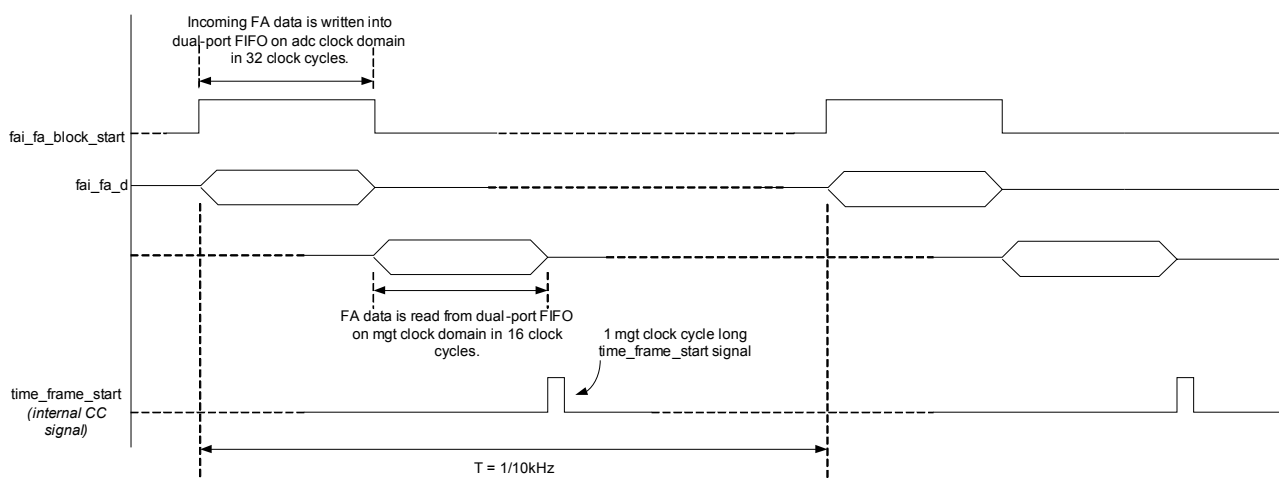


Figure 8. Fast Acquisition Data Transfer Timing.

Due to nature of data transfer, it introduces a delay of:

$$T_{DELAY} = 32 \times T_{ADC} + 14 \times T_{FOFB} \approx 500 \text{ ns.}$$

The *fofb_cc_fa_limiter.vhd* module includes the logic in order to set a limit on the number of FA data packets to be accepted by the communication controller. This functionality is

used for debugging of the core. The limit is set using bits 32-4 of *fai_cfg_val* input. For continuous operation this value should be set to 0.

fofb_cc_frame_cntrl.vhd module handles different *time_frame_start* inputs. It accepts *time_frame_start* signal from Libera core through *fa_interface* module and *time_frame_start_bits* detected from incoming packets. Module, then, outputs one *tfs_out_cc* pulse for internal operation and do not accept any incoming pulses within the time frame period length set by user.

4.1.3 BPM Communication Controller

4.1.3.1 Arbiter/Multiplexer (ArbMux) Module (*fofb_cc_arbmux.vhd*)

```
entity fofb_cc_arbmux is
port (
    -- mgt_clk
    mgt_clk          : in std_logic;
    -- System Reset (Active high)
    mgt_rst          : in std_logic;
    -- Input Data structure (RioCount x 128-bit wide)
    data_in          : in std_logic_2d_128(RioCount-1 downto 0);
    -- DataRdy in for each channel
    data_in_rdy      : in std_logic_vector(RioCount-1 downto 0);
    -- Read enable output to RX FIFO
    rx_fifo_rd_en    : out std_logic_vector(RioCount-1 downto 0);
    -- Control ports
    channel_up       : in std_logic_vector(RioCount-1 downto 0);
    -- Output data (1 x (32*PacketSize)-bit wide)
    data_out         : out std_logic_vector((32*PacketSize-1) downto 0);
    -- Output Data ready signal. Active for 1 cc during the first word of packet.
    data_out_rdy     : out std_logic;
    -- time frame finished
    frame_time_out   : in std_logic;
    -- time frame start
    frame_start      : in std_logic
);
end fofb_cc_arbmux;
```

Figure 9. ArbMux module VHDL entity declaration.

mgt_clk : CC system clock rate, 106.25MHz.

mgt_rst : MGT domain reset.

data_in : 2-D input array of 128-bits to ArbMux module from RX fifos.

data_in_rdy : Data ready pulse from each RX fifo. The arbmux module monitors this input to retrieve data from each fifo. Each bit in the array corresponds to an input fifo.

rx_fifo_rd_en : Read enable signal to input fifos. Each bit in the array corresponds to an input queue.

channel_up : This signal indicates if the RocketIO channel is active for data transmission. Each bit in the array corresponds to an input queue.

data_out : Output data to fofb_cc_fod.vhd module. The incoming packets from all queues are retrieved by a round-robin algorithm and send to Forward or Discard module for processing.

data_out_rdy : Data ready pulse for output data.

frame_time_out : Frame timeout input reset the arbmux state machine into idle state.

frame_start : Frame start pulse triggers the start of arbmux module operation.

4.1.3.2 Implementation Details

Outputs of the RX fifos are fed in to Arbiter/Multiplexer (ArbMux) module. The data path from each channel at the output of the RX fifo is 128-bit wide. The ArbMux module consists of a simple state machine implementing the round-robin algorithm in order to retrieve queued packets from all incoming channels. Each channel is assigned four time slots as long as there is data queued in the channel fifo. The ActiveChannel variable keeps the channel number where data is being read. If the fifo for the ActiveChannel is empty, ActiveChannel variable is incremented by one to retrieve data from the next available channel.

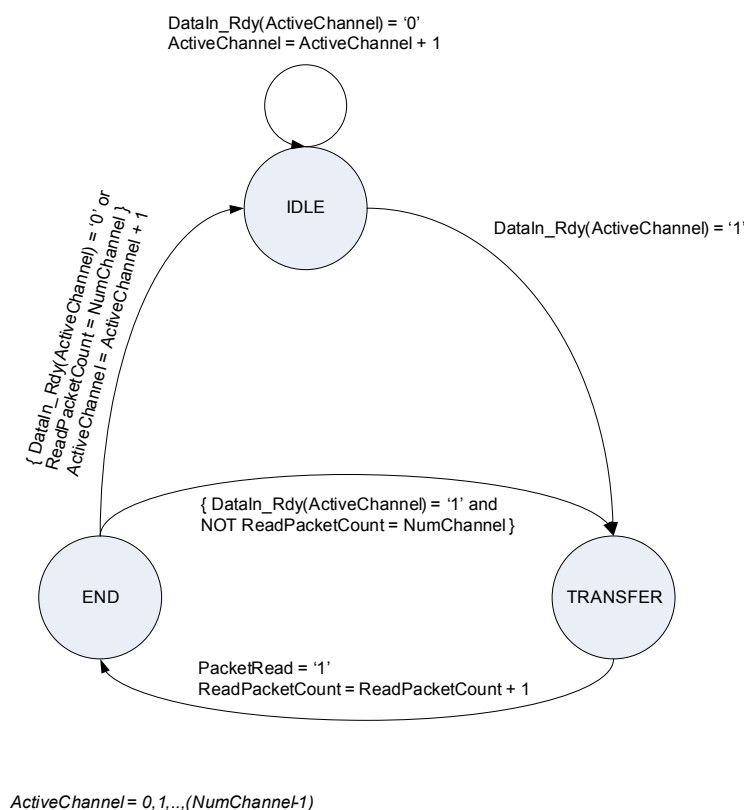


Figure 10. ArbMux state machine.

4.1.3.3 Forward or Discard (FoD) Module (fofb_cc_fod.vhd)

```

entity fofb_cc_fod is
port (
    mgt_clk          : in std_logic;
    sys_clk          : in std_logic;
    mgt_rst          : in std_logic;
    -- Frame start (1cc long)
    frame_start      : in std_logic;
    -- Channel up between channels
    channel_up       : in std_logic_vector(RioCount-1 downto 0);
    -- Incoming data for arbmux to be Forwarded or Discarded
    fod_data_in      : in std_logic_vector((PacketSize*32-1) downto 0);
    fod_data_in_rdy   : in std_logic;
    -- Injected or Forwarded Data
    fod_data_out      : out std_logic_vector((PacketSize*32-1) downto 0);
    Fod_data_out_wen  : out std_logic_vector(RioCount-1 downto 0);
    -- Frame status data
    time_frame_cnt    : in std_logic_vector(31 downto 0);    -- Time frame number.
    frame_time_out    : in std_logic;                        -- Time frame is finished.
    -- Packet information coming from Libera interface
    bpm_x_pos        : in std_logic_vector(31 downto 0);
    bpm_y_pos        : in std_logic_vector(31 downto 0);
    -- Dummy/Real position data select
    pos_data_sel_i    : in std_logic;
    -- TX FIFO full status
    tx_fifo_full      : in std_logic_vector(RioCount-1 downto 0);
    -- CC Configuration registers
    bpm_id           : in std_logic_vector(9 downto 0);
    -- X and Y pos out to CEP and PMC Interface
    pos_dout         : out std_logic_vector(31 downto 0);
    pos_addr         : in std_logic_vector(9 downto 0);
    -- status info
    fod_process_time  : out std_logic_vector(15 downto 0);
    bpm_count        : out std_logic_vector(7 downto 0);
    -- golden orbit
    golden_orb_x      : in std_logic_vector(31 downto 0);
    golden_orb_y      : in std_logic_vector(31 downto 0);
    -- fofb system heartbeat and event (go, stop) signals
    fofb_heart_beat   : in std_logic;                        -- FOFB heart beat
    fofb_event        : in std_logic_vector(15 downto 0)     -- FOFB Go flag
);
end entity;

```

Figure 11. FoD module VHDL entity declaration.

mgt_clk : MGT domain clock rate.

sys_clk : 60 MHz system clock for PMC design.

mgt_rst : MGT domain reset.

frame_start : frame_start pulse starts a new timeframe. With the rising edge of this signal, FOD module registers its own BPM values (X position, Y position) on its ports to be injected to the CC network.

fod_data_in : Input data from ArbMux module to be processed. The data is fully parallelised as 128-bits.

fod_data_in_rdy: Data ready pulse. The length of the pulse is 1 clock cycle. The FOD module registers input data with the rising edge of this pulse for further processing.

fod_data_out : Fully parallelised output packet to be written to output queue of all RocketIO channels. Each RocketIO channel has its own queue. Same output data is fed into all queues.

fod_data_out_wen : Write enable signal in order to write output packet to Transmit (output) fifo. Writing to output queues are managed individually. This signal is generated according to TXF_Full signal. If output queue is full of a particular channel, no data write takes place.

time_frame_cnt : Time frame number to be embedded into injected BPM header. The counter is implemented on higher level design entity and its value is send to FOD module.

frame_time_out : Indicates timeout for the current time frame.

bpm_x_pos :BPM's own X position value.

bpm_y_pos :BPM's own Y position value.

pos_data_sel_i : Select position data to be placed in the payload. 0 : read position data. 1 : synthetically generated position data for debugging purposes.

tx_fifo_full : Fifo full flag from each TX fifo. If a TX fifo is full, data is not written.

bpm_id : Bpm_id value from config_interface to be written in the packet header.

fod_process_time : BPM data distribution time (including PMC packets) for each time frame.

bpm_count : Number of nodes (BPM + PMC) on the communication network.

4.1.3.4 Implementation Details

The Forward or Discard (FoD) module is responsible for:

- injection of the BPM's own payload at the beginning of each time frame,
- distributing the received BPM payloads,
- extracting and storing position values.

These three functions are coordinated by the control state machine in the module.

Injection of the BPM's own payload:

The control state machine is initially in *idle* state waiting for the time frame to start. When it receives 1 clock cycle period *frame_start* pulse, it puts the BPM's parallelised 4x32-bit own payload on the *fod_data_out* bus and asserts *fod_data_out_wen* signal for 1 clock cycle in order to write the payload data to the TX fifos, and then goes to *forward_idle* state.

TimeFrameStartBit is also set to '1' in the header field of the injected packet.

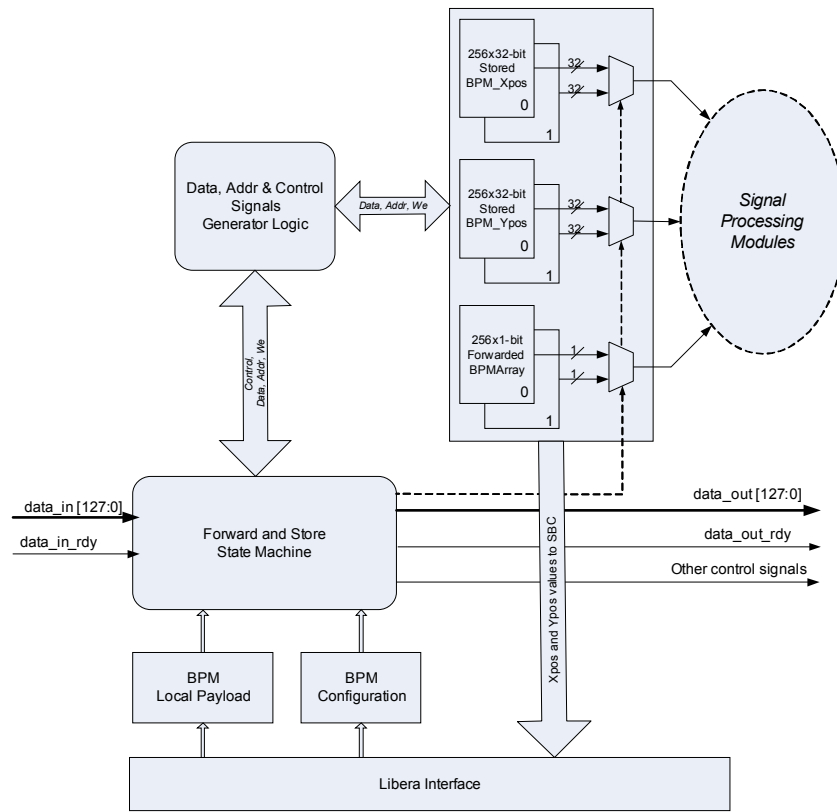


Figure 12. FoD module architectural block diagram.

Distributing the received BPM payloads:

In *forward_idle* state, the control state machine waits for arbitrated incoming packets from ArbMux module. The ArbMux module asserts *fod_data_in_rdy* signal for 1 clock cycle period when a new packet is available on the *fod_data_in* bus. With the receipt of new packet, the BPM ID extracted from the received packet is placed on the address bus of the ForwardedBitArray look-up table, and then control state machine goes to *forward_check* state in order to check if this BPM value has already been received and processed. A '1' value on the output data bus of the ForwardedBitArray look-up table indicates that a packet has already been received from the same BPM, therefore the packet is discarded. A '0' indicates that it is the first time that a packet is received from this particular BPM in this time frame, and the packet is written to all TX fifos (forwarding) to be transmitted by asserting *fod_data_out_wen* output and '1' is written to the corresponding memory location of the ForwardedBitArray look-up table. During forwarding process, *TimeFrameStartBit* located in the header field of the packet is cleared. The control state machine goes back to *forward_idle* state to receive further incoming packets. In this state, *frame_time_out* pulse forces control state machine to go back to *idle* state and wait for the next time frame start.

Extracting and storing position values:

To store a 32-bit x and y position value from all 168 BPMs around the storage ring, two 168 x 32-bit BlockRAMs are required¹. Memory locations to store position values are allocated according to BPM IDs. Additionally, each BRAM has been double buffered in order to provide continuous operation.

BPM's own position values are written to position BRAMs in parallel to injection of the BPM's own payload in the *idle* state of the control state machine. Position values of the other BPMs are written in the *forward_check* state, when a packet is received from a particular BPM for the first time.

The buffers are switched in the beginning of each time frame. *DB_Mode* signal controls the input and output multiplexers for all those three double buffers.

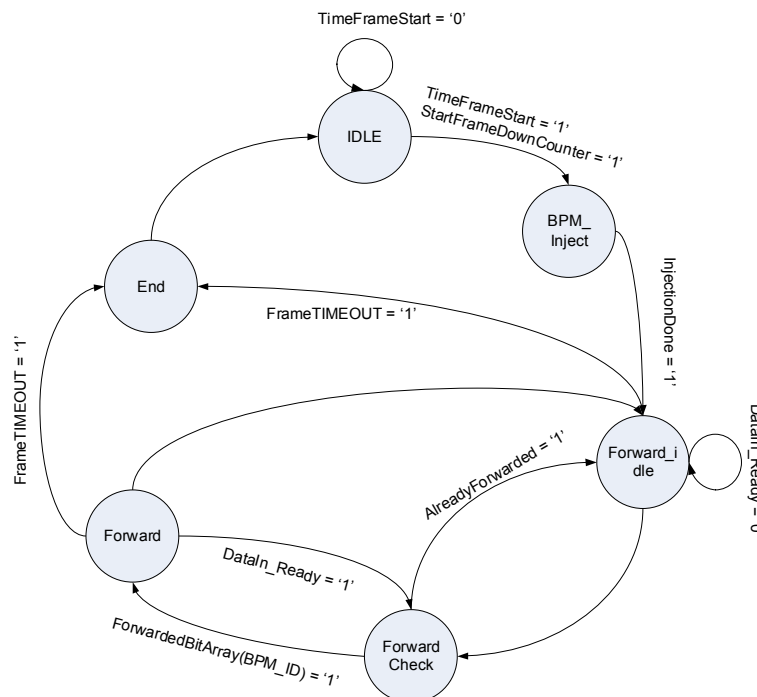


Figure 13. Forward or Discard Module State Machine.

4.1.3.5 TX and TX fifos (fofb_cc_rx/tx_fifo.ngc)

1-to-8 and 8-to-1 asymmetric fifos are implemented for RX and TX fifos, respectively, in order to handle datapath length conversion between communication controller processing modules and RocketIO transceiver modules. The fifo depth is set to 32 128bit-words.

¹ In the current version of the CC design, 256x32-bit double buffers are implemented for future use.



The fifos are generated using Xilinx coregent tool and are kept as netlist files in the directory structure.

4.1.4 Rocket IO Interface Module (*fofb_cc_mgt_interface.vhd*)

```

entity fofb_cc_mgt_interface is
  port (
    -- clocks and resets
    bref_clk      : in  std_logic;
    user_clk      : in  std_logic;
    mgt_rst       : in  std_logic;
    -- RocketIO
    rxn_in        : in  std_logic;
    rxp_in        : in  std_logic;
    txn_out       : out std_logic;
    txp_out       : out std_logic;
    -- time frame sync
    frame_timeout : in  std_logic;
    time_frame_cnt : in  std_logic_vector(15 downto 0);
    bpm_id        : in  std_logic_vector(7 downto 0);
    --
    powerdown     : in  std_logic;
    loopback      : in  std_logic_vector(1 downto 0);
    -- status information
    link_ok       : out std_logic_vector(1 downto 0);
    frame_error   : out std_logic;
    soft_error    : out std_logic;
    tx_hard_error : out std_logic;
    rx_hard_error : out std_logic;
    rx_data_monitor : out std_logic_vector(15 downto 0);
    tx_pck_cnt    : out std_logic_vector(15 downto 0);
    rx_pck_cnt    : out std_logic_vector(15 downto 0);
    -- network information
    tfs_bit       : out std_logic;
    link_partner  : out std_logic_vector(9 downto 0);
    pnc_tf_cnt    : out std_logic_vector(15 downto 0);
    -- tx/rx state machine status for reset operation
    tx_sm_sta     : out std_logic;
    rx_sm_sta     : out std_logic;
    -- TX FIFO interface
    tx_data_in    : in  std_logic_vector(15 downto 0);
    tx_data_rdy   : in  std_logic;
    txf_rd_en_i   : out std_logic;
    -- RX FIFO interface
    rx_data_out   : out std_logic_vector(15 downto 0);
    rx_data_rdy   : out std_logic;
  );
end fofb_cc_mgt_interface;

```

Figure 14. MGT Interface module VHDL entity declaration.

bref_clk : Dedicated 106.25 MHz clock input to RocketIO transceiver.

user_clk : 106.25 MHz clock for communication controller operation.

mgt_rst : Mgt clock domain reset.

rxn_in : Negative differential serial data input pin.

rxp_in : Positive differential serial data input pin.

txn_out : Negative differential serial data output pin.

txp_out : Positive differential serial data output pin.

frame_timeout : Time frame timeout input. This signal resets TX and RX state machines into *idle* state.

time_frame_cnt : Time frame number input. An incoming packet is discarded if its *time_frame_count* value does not match this input.

bpm_id : bpm number value to be send to link partners.

powerdown : *Powerdown* input from *fofb_cc_config_intf* module.

loopback : *Loopback* input from *fofb_cc_config_intf* module.

link_ok :

frame_error : Asserted for 1 clock cycle when a frame error is detected.

soft_error : Asserted for 1 clock cycle when a soft error is detected.

tx_hard_error : Asserted for 1 clock cycle when a hard error on TX side is detected.

rx_hard_error : Asserted for 1 clock cycle when a hard error on RX side is detected.

rx_data_monitor : Not used.

tx_pck_cnt : Number of transmitted packets.

rx_pck_cnt : Number of received packets.

tf_s_bit : time frame start bit value extracted from incoming packets. This signal is used to indicate start of time frame in the CC design on PMC.

link_partner : *bpm_id* value of the link partner.

pmc_tf_cnt : Time_frame number of the first incoming packet in each time frame. This value is used in PMC CC for synchronisation.

tx_sm_sta : TX state machine status. If this value is 1, packet transmission is ongoing and TX fifos should not be reset.

rx_sm_sta : RX state machine status. If this value is 1, packet reception is ongoing and RX fifos should not be reset.

tx_data_in : BPM packet data from TX fifo to be transmitted over the channel.

tx_data_rdy : TX fifo full flag. This signal indicates that there is at least one BPM payload is available in the fifo.

txf_rd_en_i : Read data form TX fifo.

rx_data_out : Output to RX fifo.

rx_data_rdy : Write enable output to RX fifo.



4.1.4.1 Implementation Details

fofb_cc_mgt_interface module implements:

- Xilinx RocketIO Transceiver Core Interface
- RX State Machine
- TX State Machine
- Error detection logic

There are eight RocketIO transceivers available on the Virtex2Pro30 FPGA device. Four of them are located on the bottom edge of the FPGA while the rest 4 MGTs are located on the top edge. In the Communication Controller design, four RocketIO transceivers located at the bottom edge are being used.

Xilinx RocketIO core is configured as:

- 106.25Mhz BREFCLK is the reference clock generated from an external source.
- Serial speeds of 2.12 Gb/s.
- 16-bit data interface.
- User Mode CRC with following parameters:
 - o CRC_END_OF_PKT = "K29_7";
 - o CRC_START_OF_PKT = "K27_7";

The following four types of data are transmitted over a Rocket IO channel.

Clock Compensation Sequence : Sequences of clock compensation symbols are used to prevent overrun of the receiver due to differences in the clock rate between channel partners. The clock compensation sequence used in CC is /K28.5/K28.7/.

Idle Sequence : Sequences of IDLE symbols are transmitted over channel whenever there is no data to send. IDLE sequence used in CC is /K28.5/K28.7/.

Position Data Packets : Position data payload is encapsulated by /SOP/ and /EOP/ symbols. /SOP/ sequence symbols used in CC are /K28.2/K27.7/ and /EOP/ sequence symbols are /K29.7/K30.7/

Bpm ID sequence : BPM ID of a channel partner is transmitted over the channel every half period ~ 50usec. The BPM ID sequence /K23.7/ is accompanied by the bpm_id value on the least significant byte of the 16-bit data interface.

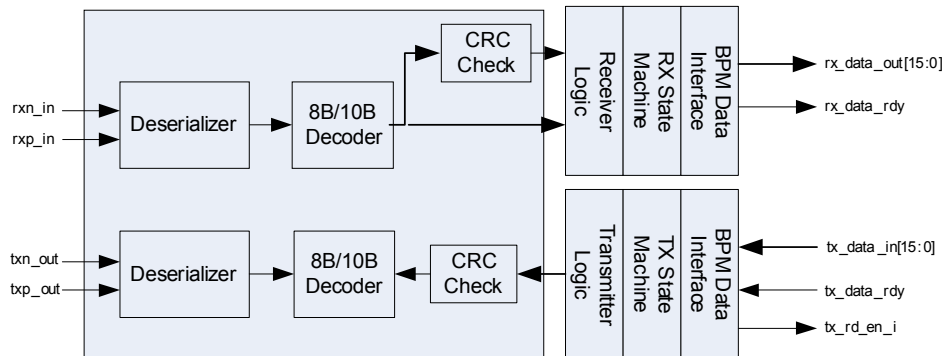


Figure 15. CC Receiver Module

Data Transmission Procedure

TX state machine handles channel initialisation and data transmission.

On power-up, reset or channel failure, state machine goes to *tx_rst* state where *tx_reset* input of the RocketIO core is asserted for 7 clock cycles for a proper reset operation. In *tx_sync* state, IDLE symbol is transmitted for *TX_NUM_IDLE* clock cycles for synchronisation with the RX side of the link partner.

In order to transmit data, the TX state machine asserts module's *txf_rd_en_i* in *tx_idle* state, when *tx_data_rdy* input is asserted by TX fifo control logic which indicates that there is a payload in the fifo ready to be sent. The encapsulation of the payload starts with the insertion of /SOP/ control symbol sequence in *tx_sop* state. Then, 4x32-bit BPM payload is accepted through *tx_data_in* port of the module. The TX state machine reads 16-byte payload from TX fifo as 8 blocks of 16-bit data and inserts a 4-byte dummy data² and a four-byte placeholder for the CRC. Finally encapsulation of the payload is finished by inserting /EOP/ control symbol sequence. The resulting data structure is referred as the *link layer payload*. The link layer payload is 8B/10B encoded and then transmitted by RocketIO transceiver through the channel.

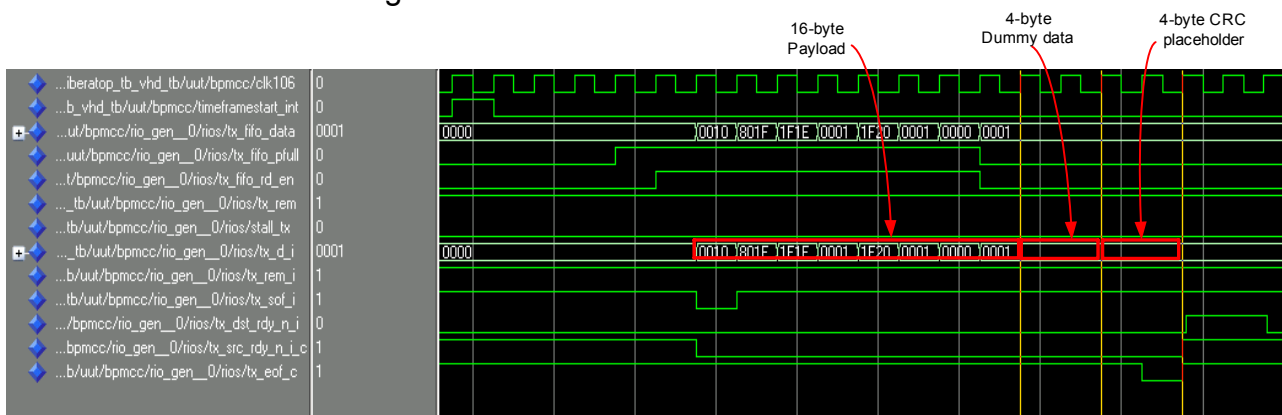
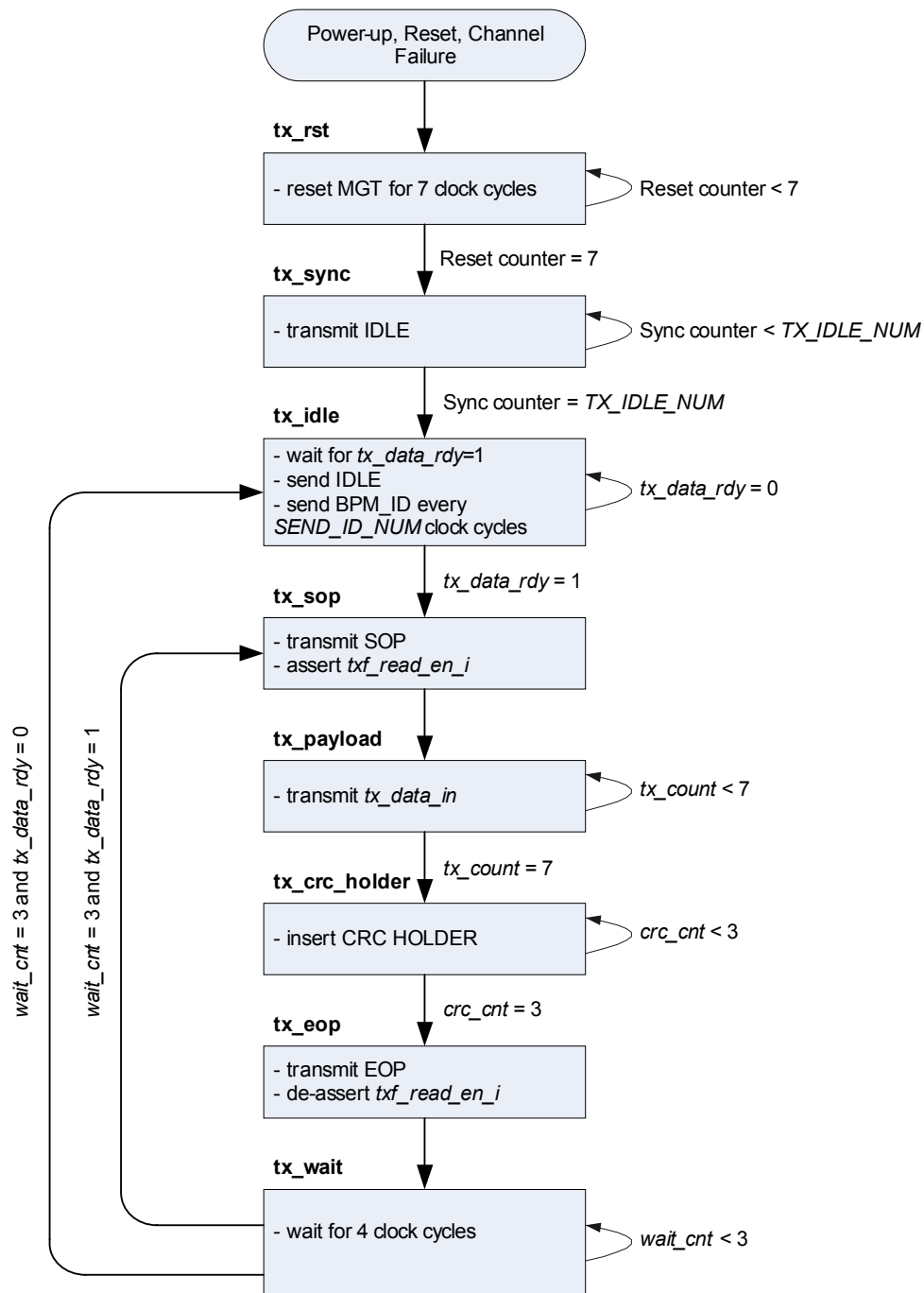


Figure 16. Payload transmission timing on CC.

² Data length must be greater than 20 bytes to RocketIO core for CRC generation. Therefore, a 4-byte dummy data has to be added at the tail of 16-byte user payload.

A state diagram of the TX state machine operation is shown below.



Data Reception Procedure

When RocketI/O MGT receives a packet through its RX channel, it deserialises this stream into 10-bit data and control symbols which is called link layer payload. After deserialisation, the link layer payload is decoded into a stream of octets and presented on core's *rx_data* output port.

To receive data, the *RX state machine* monitors the SOP control symbol sequence on the *rx_data* output port of the RocketIO core. When valid data is present on the *rx_data* port, the RX state machine starts reading payload data. It removes 4-byte dummy data and 4-byte CRC data before writing 16-byte payload to the RX fifo. After link layer payload stripping, the 4x32-bit BPM payload is written into asymmetrical RX fifo as 8 blocks of 16-bit data, or

If a CRC error occurs or the time frame number value of the received data does not match with the local BPM's time frame value, the payload is discarded without being written to Rx fifo.

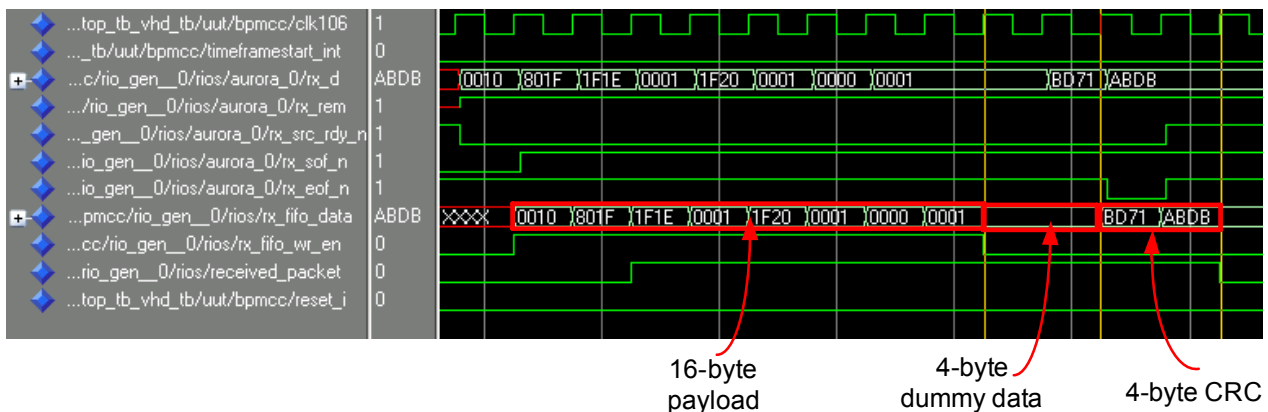
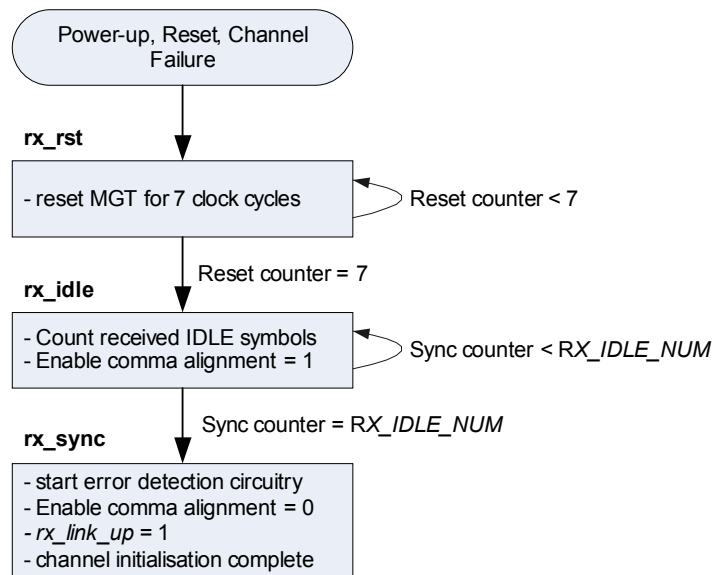


Figure 17. Payload receiver timing on CC.

RX Channel Initialisation

On power-up, reset or channel failure, *rx_init* state machine handles simplex channel initialisation as shown in the state diagram below. The state machine asserts *rx_link_up* when pre-defined numbers of IDLE symbols are received consecutively over the channel.



Error Detection

The module reports CRC errors on incoming packets by asserting *frame_error* signal for 1 clock cycle.

soft_error signal is asserted when the 10-bit code received from the channel partner is not a valid code in the 8B/10B table, or when the code does not have the correct disparity.

The error detection logic monitors RocketIO transceiver for hardware errors such as buffer overflow and loss of lock. Hard errors also shows themselves as burst of soft errors. The module reports hardware errors by asserting the *hard_error* signal.

Whenever a hard error detected, the error detection logic resets TX and RX channels accordingly.

PMC operation

The PMC modules do not have access to global synchronised *time_frame_start* signal which indicates the start of a time frame. Therefore, communication controller on PMC modules is able to detect the start of a time frame by extracting the *time_frame_start_bit* from an incoming packet header.

time_frame_start_bit is embedded in the most 16-bit word of the 32-bit header. To detect most significant 16-bit word of the incoming packet header is delayed by one clock cycle.

4.2 CC Design on the PMC Modules

The CC design for the PMC will have a similar architecture as the CC design for the BPM. The main differences are:

- The PMC will not generate own BPM values.
- In addition to the BPM CC functionality, it stores the values in a double buffer to make the BPM values available to the fast feedback processor. The buffers are switched when a new time frame starts.
- Because the PMC has no direct connection to the timing system, it has to detect the beginning of a new time frame. This is signalled by a flag in the first packet of a time frame coming from a BPM.
- A time frame timeout counter will generate an interrupt after a configurable time after a time frame start signal has been received. This is to reduce latency between receiving the last BPM value and notifying the software. The duration shall be configurable up to 250µs in steps of microseconds.

4.3 Compile time user definitions

Compile time options for the CC design must be correctly defined for correct design compilation according to accelerator needs. The options below are found in *fofb_cc_user_defines.vhd* file.

FA_CAPTURE : Enable/disable capture of 1024 samples of x and y position data (at FA rate) on the external trigger. The captured data is stored until read by SBC interface.

EXTENDED_CONF_BUF : Sections of configuration BRAM which hold inverse response matrix coefficients are transferred into individual buffers (*coeff_x* and *coeff_y*) so that they can be used in feedback computation part of the design.

TX_BPM_POS_ABS : Sets the way that position values are placed in the payload. Transfer position data as read through FA interface (true), or subtract *golden_orbit* value from FA data value (false).

DATA_STORE_EN : Enable/disable on-chip storing of full set of x&y position data. Used in PMC design.

4.4 X and Y position array read interface

If **DATA_STORE_EN** compile time option is set to *true*, full set of x and y position data from all nodes on the communication network are stored on double buffered arrays. The buffers are named as *XPosMem_A/B* and *YPosMem_A/B* in *fofb_cc_fod.vhd* file. They are implemented using 32-bitx256 BRAMs.

Stored set of x and y position arrays can be read (in parallel) using following position data interface ports on the top-level design (*fofb_cc_top*).

Position data interface			
frame_timeout	Out	1	Frame timeout pulse
pos_addr	In	11	Position data address
pos_dout	Out	32	Position data value

The timing of the position data interface is shown in Figure 18. The internal buffers are switched at FA rate (10.072 kHz for Diamond), therefore external application has to finish reading x and y position buffers within $(1/f_a_rate)$ μ sec. The data reading procedure can be summarised in two steps:

- 1-) User application logic should monitor rising edge of *frame_timeout* signal. This signal indicates that current time frame ended and x and y position arrays are ready to be read.
- 2-) Start reading position arrays using *pos_addr* and *pos_dout* ports. There is a 1 clock cycle latency to data output.

Please note that position read interface should be handled in sys_clk domain.

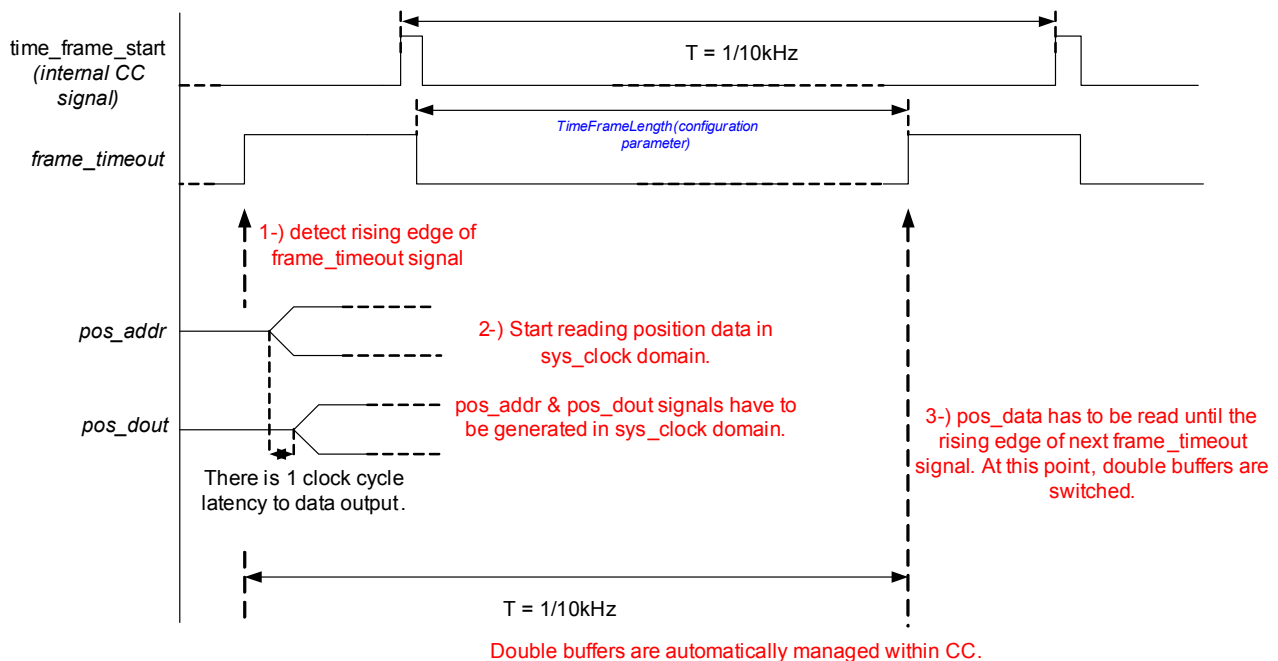


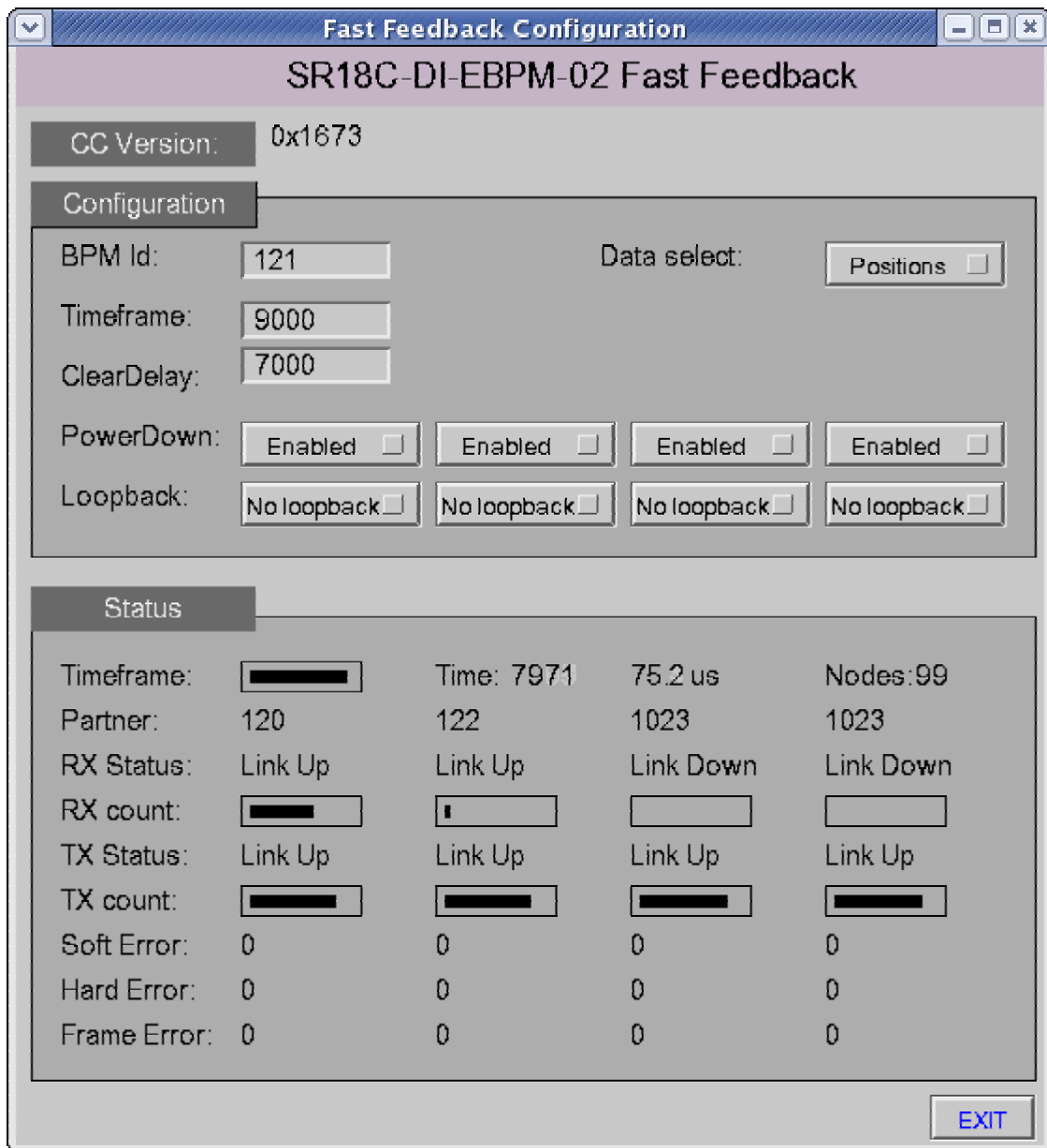
Figure 18 Position data interface timing.

5. COMMUNICATION CONTROLLER MANAGEMENT

A high-level EPICS application has been developed to configure and monitor status of CC operation on BPMs.

The application provides a user interface to:

- Configure CC on BPMs
- Display status information hierarchically (top-down displays)
- On-demand display and check for correct wiring of BPMs (through Matlab).



Fast Feedback Configuration

SR18C-DI-EBPM-02 Fast Feedback

CC Version: 0x1673

Configuration

BPM Id: 121 Data select: Positions ☐

Timeframe: 9000

ClearDelay: 7000

PowerDown: Enabled ☐ Enabled ☐ Enabled ☐ Enabled ☐

Loopback: No loopback ☐ No loopback ☐ No loopback ☐ No loopback ☐

Status

Timeframe:		Time: 7971	75.2 us	Nodes: 99
Partner:	120	122	1023	1023
RX Status:	Link Up	Link Up	Link Down	Link Down
RX count:				
TX Status:	Link Up	Link Up	Link Up	Link Up
TX count:				
Soft Error:	0	0	0	0
Hard Error:	0	0	0	0
Frame Error:	0	0	0	0

EXIT

Figure 19 EPICS user interface for Fast Feedback Configuration on BPM

5.1 Link Status Query and Fault Monitoring

The status of a receiving link will be queried by accessing registers of a CC through the EPICS interface. The status consists of:

- the information to which BPM or PMC the receiving link is connected to and if the link is working, i.e. locked,
- number of nodes of the communication network,

- data distribution time over communication network,
- number of transmitted and received packet count over each channel,
- number of link errors on each channel,
- number of lost packets and,
- link up time (TBD).

5.2 Test Functionality

The test mode, to be implemented as part of current CC design, shall be controlled by EPICS interface. In this mode, a predefined position data will be injected to the network by each BPM and CC operation will be monitored and verified through high-level user application.

6. DESIGN INTERFACE AND INTEGRATION

In order to instantiate Communication Controller in the top-level FPGA design, following code blocks should be added into the top-level design file.

1-) Instantiate differential input clock pad.

Reference clock for the Rocket IOs is supplied by an external 106.25MHz clock oscillator with differential output.

```
//=====
//
// 106MHz clock signal fed through differential input pad
//
//=====

wire    clk106_ipad ;
IBUFGDS_LVPECL_25 i_IBUFGDS_LVPECL_25_clk106
(
    .I (    clk106_ppad_i    ),
    .IB (    clk106_npad_i    ),
    .O (    clk106_ipad      )
);
```



2-) Instantiate RocketIO DCM and user clock buffer.

A DCM is used to generate user_clock derived from 106.25MHz clock input.

```
//=====
//
// RocketIO DCM and Clock buffer instantiation
//
//=====

wire      user_clk ;
wire      mgt_dcm_locked;
wire      mgt_dcm_clk0;

DCM i_DCM_mgt
(
    .CLK0      (    mgt_dcm_clk0      ),
    .CLK180    (                      ),
    .CLK270    (                      ),
    .CLK2X     (                      ),
    .CLK2X180  (                      ),
    .CLK90     (                      ),
    .CLKDV     (                      ),
    .CLKFX     (                      ),
    .CLKFX180  (                      ),
    .LOCKED    (    mgt_dcm_locked    ),
    .PSDONE    (                      ),
    .STATUS    (                      ),
    .CLKFB     (    user_clk          ),
    .CLKIN     (    clk106_ipad       ),
    .DSSSEN    (    1'b0              ),
    .PSCLK     (    1'b0              ),
    .PSEN      (    1'b0              ),
    .PSINCDEC  (    1'b0              ),
    .RST       (    !dcm_main_rst     )
);

BUFGMUX i_BUFGMUX_mgt
(
    .IO (    mgt_dcm_clk0    ),
    .I1 (    1'b0           ),
    .O  (    user_clk       ),
    .S  (    1'b0           )
);
```



3-) Implement reset logic for mgt domain.

```
///=====
// RocketIO clock domain reset
//=====

wire    mgt_rst_so ;
wire    mgt_rst;

SRL16 i_SRL16_gmii_rst
(
    .Q      (    mgt_rst_so          ),
    .A0     (    1'b1                ),
    .A1     (    1'b1                ),
    .A2     (    1'b1                ),
    .A3     (    1'b1                ),
    .CLK     (    user_clk            ),
    .D      (    mgt_dcm_locked      )
);
always@(posedge user_clk)
begin
    mgt_rst    <= mgt_rst_so ;
end
```

4-) Instantiate communication controller

```
///=====
// Fast Feedback Modules
//=====

wire [3:0] mgt_rxp;
wire [3:0] mgt_rxn;
wire [3:0] mgt_txp;
wire [3:0] mgt_txn;

assign mgt_rxp[0]    = rio1_rdp_pad_i ;
assign mgt_rxn[0]    = rio1_rdn_pad_i ;
assign mgt_rxp[1]    = rio2_rdp_pad_i ;
assign mgt_rxn[1]    = rio2_rdn_pad_i ;
assign mgt_rxp[2]    = rio3_rdp_pad_i ;
assign mgt_rxn[2]    = rio3_rdn_pad_i ;
assign mgt_rxp[3]    = rio4_rdp_pad_i ;
assign mgt_rxn[3]    = rio4_rdn_pad_i ;

assign rio1_tdp_pad_o = mgt_txp[0];
assign rio1_tdn_pad_o = mgt_txn[0];
assign rio2_tdp_pad_o = mgt_txp[1];
assign rio2_tdn_pad_o = mgt_txn[1];
assign rio3_tdp_pad_o = mgt_txp[2];
assign rio3_tdn_pad_o = mgt_txn[2];
assign rio4_tdp_pad_o = mgt_txp[3];
assign rio4_tdn_pad_o = mgt_txn[3];
```



```

fofb_cc_top i_fofb_cc_top
(
    // Clocks and resets
    .adc_clk          ( adc_clk          ),
    .mgt_clk          ( clk106_ipad      ),
    .user_clk_i       ( user_clk         ),
    .adc_rst          ( !adc_rst         ),
    .mgt_rst          ( !mgt_rst         ),
    .sys_clk          ( 1'b0             ),
    // FA rate data interface
    .fai_fa_block_start ( fai_fa_block_start ),
    .fai_fa_data_valid  ( fai_fa_data_valid ),
    .fai_fa_d           ( fai_fa_d        ),
    // FAI configuration interface
    .fai_cfg_a          ( fai_cfg_a       ),
    .fai_cfg_do          ( fai_cfg_do      ),
    .fai_cfg_di          ( fai_cfg_di      ),
    .fai_cfg_we          ( fai_cfg_we      ),
    .fai_cfg_clk         (                ),
    .fai_cfg_val         ( fai_cfg_val     ),
    // PMC mode of operation data transfer interface
    .fai_rio_tdis        (                ),
    .fai_rio_rdp         ( mgt_rxp        ),
    .fai_rio_rdn         ( mgt_rxn        ),
    .fai_rio_tdp         ( mgt_txp        ),
    .fai_rio_tdn         ( mgt_txn        ),
    // PMC mode of operation data transfer interface
    .frame_timeout      (                ),
    .pos_dout           (                ),
    .pos_addr           ( {10{1'b0}}      ),
    .coeff_x_addr       ( {8{1'b0}}       ),
    .coeff_y_addr       ( {8{1'b0}}       ),
    .coeff_x_out        (                ),
    .coeff_y_out        (                ),
    // PMC mode of operation handshaking & status signals
    .fofb_heart_beat    ( 1'b0            ),
    .fofb_event         ( {16{1'b0}}      ),
    .fofb_process_time  (                ),
    .fofb_bpm_count     (                ),
    .fofb_dma_ok        ( 1'b1            ),
    // SBC interface used for on-chip FA rate dare acquisition
    .ext_trig_in        ( lemo_trig_ipad   ),
    .sbc_clk            ( sbc_clk          ),
    .sbc_rst            ( !sbc_rst         ),
    .sbc_cs             ( sbc_cs[ `NEUTRINO_USR_3_CS_NUM] ),
    .sbc_wstb           ( sbc_wstb        ),
    .sbc_rstb           ( sbc_rstb        ),
    .sbc_adr            ( sbc_local_adr    ),
    .sbc_dat_i          ( sbc_write_dat    ),
    .sbc_dat_o          ( usr_cs3_sbc_read_dat ),
    .sbc_irq_o          (                )
);

```

In order to configure and start Communication controller, following steps are followed:

- 1-) Set Bit-4 of `fai_cfg_val` register to 1 to enable Communication Controller.
- 2-) Write your configuration values on 0x14028000 – 0x14028BFF accordingly.
- 3-) Set Bit-0 of `fai_cfg_val` register to 1 and, the 0 in order to create a rising edge.
- 4-) Communication controller starts reading configuration data
 - The values for `coeff_x` and `coeff_y` are loaded into individual BRAMs.
- 5-) `coeff_x` and `coeff_y` data becomes available via `coeff_x/y_addr` and `coeff_x/y_out` on the CC interface.
- 6-) Comm. Controller starts writing status information into status registers.

7. REFERENCES

- [1] DLS FFS Communication Specification, Supercomputing Systems, 2004.
- [2] The Libera product. <http://www.i-tech.si/products-libera.html>
- [3] Micro-Research Finland Oy, "Final design Review, Diamond Light Source, Timing System Modules", 2004.
- [4] <http://www.cselex.com/fibre-channel-cables.htm>