Register Command and Data Interface for the Reconfigurable Cluster Element

Interface Control Document

DEPARTMENT OF ENERGY STANFORD LINEAR ACCELERATOR CENTER

Document Control Sheet

Author(s):	Amedeo Perazzo, Ryan Herbst
ID:	ICD xxx-xxxx-xxx
Date:	09/05/07
Version:	0.1
Status:	Draft

Document Status Sheet

Name	Signature	Date
Amedeo Perazzo CCI, Software Manager		
Ryan Herbst CCI, Engineer		

Table of Contents

Chapter 1: Introduction

This document describes the hardware, firmware and software framework which allows the Reconfigurable Cluster Element (RCE) to access registers, send commands to and receive data from a generic Front End Board (FEB). See [RCE07] for more informations regarding the RCE. In the following the framework will be indicated as Register Command and Data Interface (RCDI).

As shown by the simplified diagram of figure 1.1, RCDI is composed of three logically separated interfaces: the register interface, used for reading and writing FEB registers, the command interface, used for sending out-of-band commands from the RCE to the FEB and the data interface, used for sending data from the FEB to the RCE.

From the RCDI perspective, the RCE acts as the master and the FEB as the slave since the RCE initiates operations and the FEB responds to these operations. A transaction is started by the RCE generating a packet which is sent to the FEB through a wire. This packet is then processed by the FEB RCDI core which will present a request for operation to the user logic either on the register or on the command interface.

If the user logic fails to reply to a register operation in a specified interval of time, a timeout condition will be forwarded by the RCDI core back to the RCE. While it's mandatory for the user logic to reply to requests on the register interface, the user logic may, or may not, reply to requests on the command interface. When replies are needed the user logic will use the data interface to send them. The RCDI core will process the user logic reply and will send a packet back to the RCE in order to complete the transaction.

Currently RCDI uses the Pretty Good Protocol (PGP) to exchange frames between the RCE and FEB. However, as RCDI is protocol independent, knowledge of PGP is not needed to use RCDI. See [PGP07] for a description of PGP.

Chapters 2 and 3 of this document define, respectively, the requirements of the FEB and the firmware interface on the FEB side. These two chapters are the only chapters needed to implement the slave platform. Chapter 4 defines the software interface on the RCE side. Finally, chapter 5 shows the format of the messages used in the communication protocol.

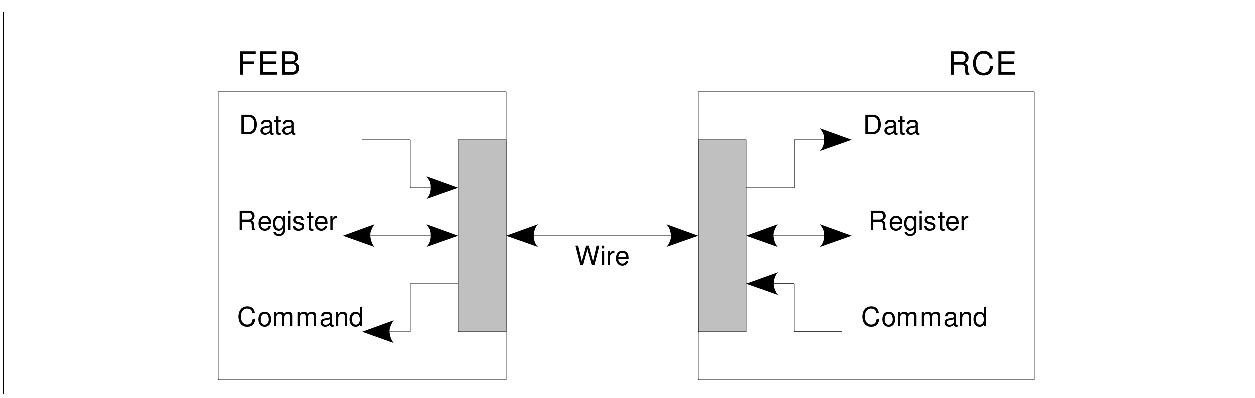

Figure 1.1: Logical diagram of the RCD interface between the RCE and the FEB.

Chapter 2: Front End Board Requirements

A simplified diagram of the physical interface between the FEB and the RCE is shown in figure 2.1.

2.1 Device Requirements

Currently the implementation of the RCDI IP core is based on the Virtex-4 FX family of devices from Xilinx. Specifically, this implementation will require the use of one or more RocketIO Multi-Gigabit Transceivers (MGT). See [VX406] for more informations about the FX family and [MGT06] for a description of the MGT.

2.2 Optical Requirements

The RCDI framework uses Avago Technologies' AFBR-59R5LZ optical transceivers. These transceivers support high-speed serial links over multi and single mode optical fibers at rates up to 4.25 Gb/s using LC connectors. One RCE board will provide up to eight full duplex transceivers (four per element) using L-com Duplex Multimode Fiber Optic Cables, 50/125 with OFNR Jacket.

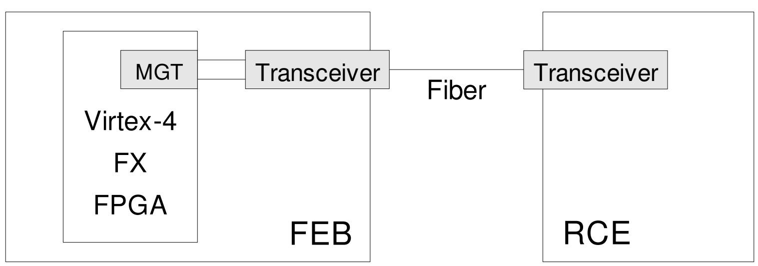

Figure 2.1: Physical diagram of the interface between RCE and FEB.

Chapter 3: Firmware Interface

This section describes the interface of the RCDI IP core with the user logic on the FEB.

The RCDI core is interfaced to the user logic through the register block, for reading and writing registers on the FEB, the command block, for sending out-of-band commands from the RCE to the FEB, and the data block, used for sending data from the FEB to the RCE, as shown in figure 3.1.

Note that each of the three interface blocks of the RCDI core is asynchronous in respect to each other. For example, data may be sent from the FEB to the RCE while the RCE is sending commands to the FEB.

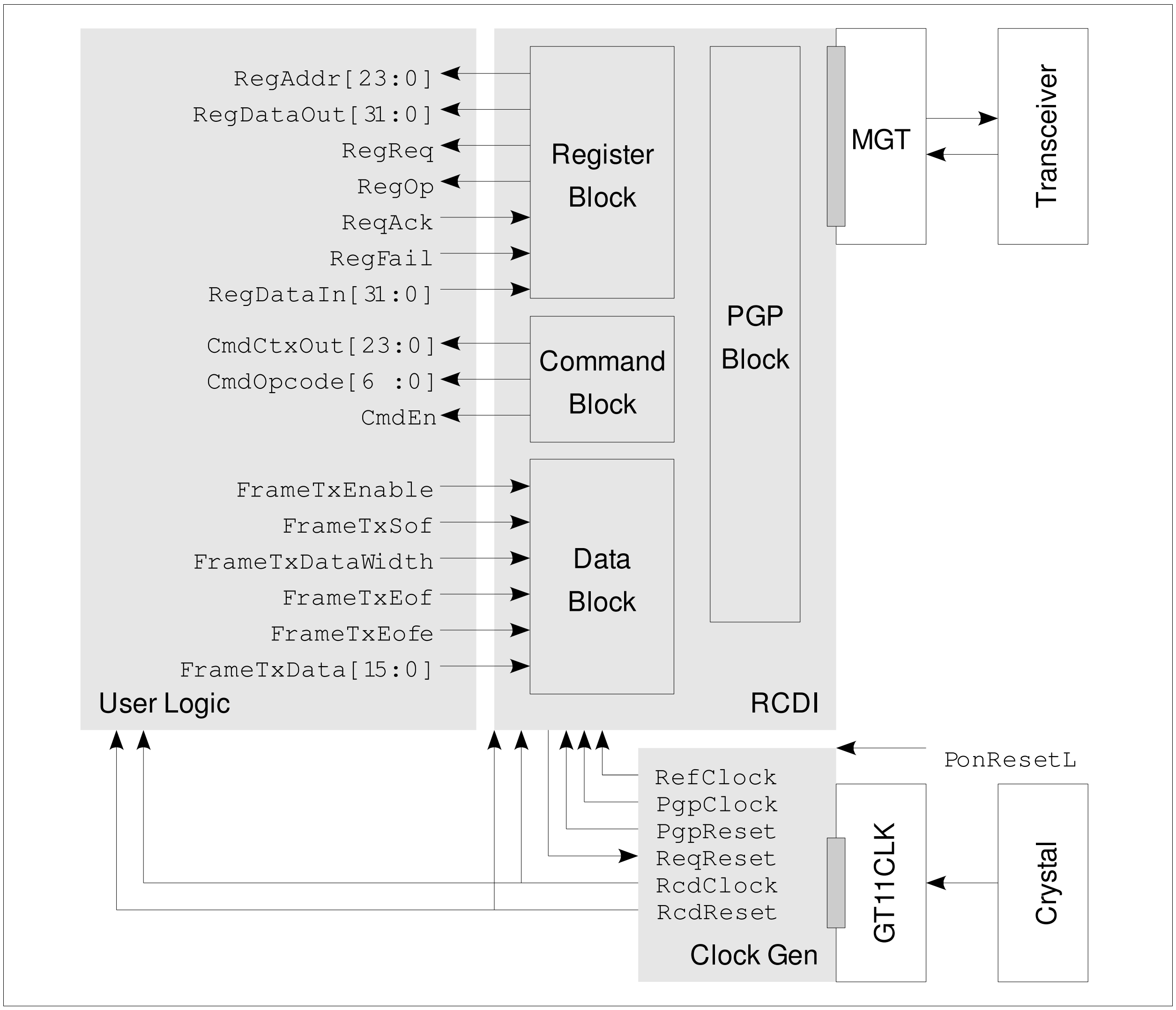

Figure 3.1: RCDI signals.

3.1 Clock Generation

The main responsibility of the clock generation block is to wrap the GT11CLK hard core and to derive from it all the clocks needed by both the RCDI core and the user logic. In addition this block also manages the reset lines for both the RCDI core and the user logic. Table 3.1 enumerates the different signals.

The RCDI core is divided in two clock domains: one is governed by `PgpClock` and is used for interfacing the core to the MGT, the other is governed by `RcdClock` and is used for interfacing the core to the user logic. The ratio between `RcdClock` and `PgpClock` may be selected in the configuration of the clock generation block, but must not be larger than 4/5, which also happens to be the default value for this ratio.

The `RefClock` signal must be generated according to the requirements defined in [MGT06]. Its frequency must be 156.25 Mhz in order to match the wire rate on the RCE side. The `PgpClock` signal has the same frequency as `RefClock` and it is derived from the latter in the clock generation block through a DCM.

The `PgpReset` and `RcdReset` lines are logically the same, but they are synchronized, respectively, with `PgpClock` and `RcdClock`. These reset lines are asserted in response to the power-on reset line `PonResetL` or in response to the `ReqReset` line from the RCDI core. The latter may be triggered by the RCE by sending a specific packet to the RCDI core.

Table 3.1: Register Interface. Direction is from the RCDI core point of view.

Signal	Width	Direction	Description
`PonResetL`	1	Input	From on-board power-on logic (active low)
`RegReset`	1	Input	Reset request line from RCDI core
`RefClock`	1	Output	Reference clock for RCDI MGT
`PgpClock`	1	Output	Clock for PGP block, same frequency as `RefClock`
`PgpReset`	1	Output	Reset line for PGP block
`RcdClock`	1	Output	Clock for RCDI interface blocks and user logic
`RcdReset`	1	Output	Reset line for RCDI interface blocks and user logic

3.2 Register Block

This block is part of the interface which allows the RCE to read/write registers from/to the FEB. Table 3.2 shows the signals between the RCDI register block and the user register block.

The RCDI register block signals the start of an operation to the user logic by asserting `RegReq`. The operation is a register write if `RegOp` is high, it's a read if `RegOp` is low.

For write operations, the user logic uses the `RegAddr` word to select which register to copy the contents of `RegDataOut`. For read operations, the user logic uses the `RegAddr` word to select which register to copy to `RegDataIn`.

The user logic acknowledges the completion of the operation by asserting RegAck. If the user logic does not acknowledge the completion within a specified time interval the register block will set the timeout bit in the reply message to the RCE. The timeout period is currently set to 127 ticks of RcdClock.

The user logic will assert RegFail together with RegAck if it was unable to complete the operation. This may happen, for example, if RegAddr does not specify a valid address.

Figures 3.2 and 3.3 show the timing diagrams of write and read operations.

Table 3.2: Register Interface. Direction is from the RCDI core point of view.

Signal	Width	Direction	Description
RegAddr[23:0]	24	Output	Address used for read and write operations
RegDataOut[31:0]	32	Output	Register value used for write operations
RegReq	1	Output	Trigger operation
RegOp	1	Output	High if operation is write, low if read
RegAck	1	Input	From user logic to acknowledge operation completion
RegFail	1	Input	From user logic to indicate failed operation
RegDataIn[31:0]	32	Input	Register value from user logic for read operations

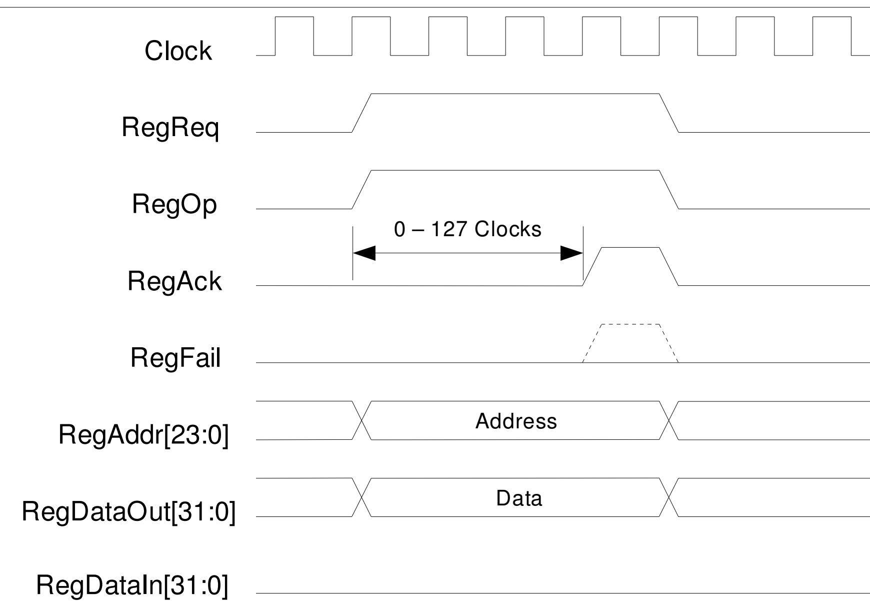

Figure 3.2: Register write timing diagram.

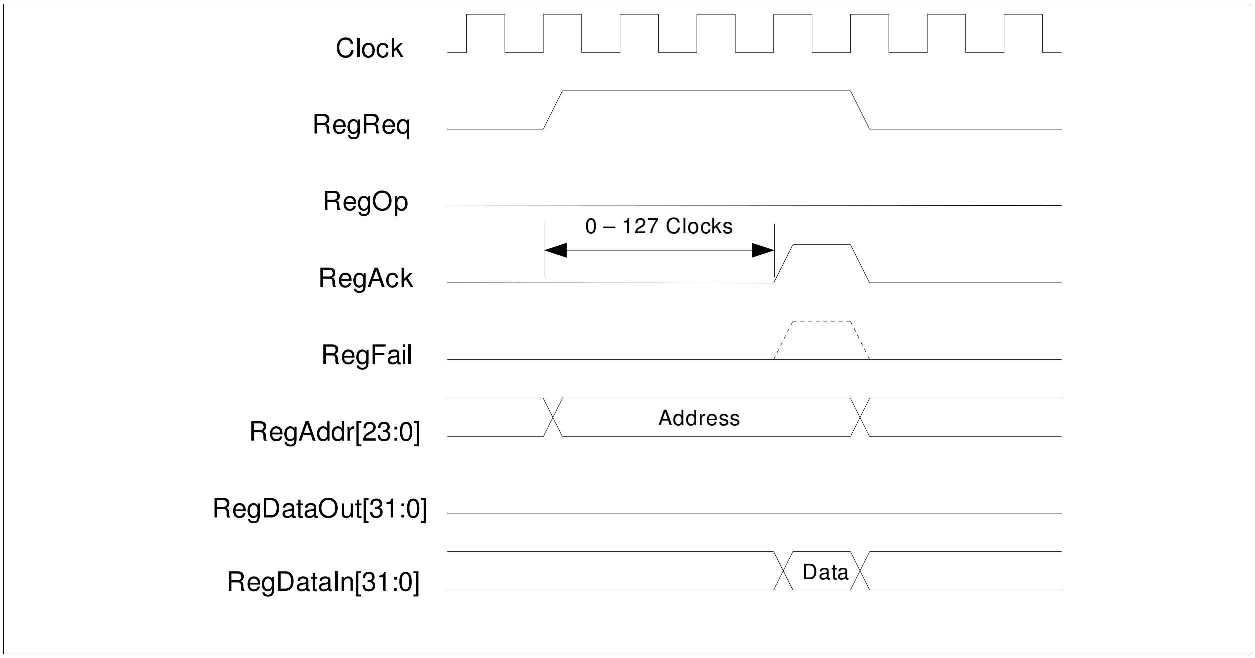

Figure 3.3: Register read timing diagram.

3.3 Command Block

This block is part of the interface which allows the RCE to send out-of-band commands to the FEB.

Table 3.3 shows the interface signals between the RCDI command block and the user command block.

A command is triggered by this block by asserting the `CmdEn` signal. As opposed from the register interface, the user logic does not acknowledge operations on the command interface.

Figure 3.4 shows the timing diagram of a command operation.

Table 3.3: Command Interface. Direction is from the RCDI core point of view.

Signal	Width	Direction	Description
`CmdCtxOut[23:0]`	24	Output	Context value:returned by user in response packet
`CmdOpcode[6 :0]`	8	Output	Opcode value
`CmdEn`	1	Output	Command is available for user logic to process

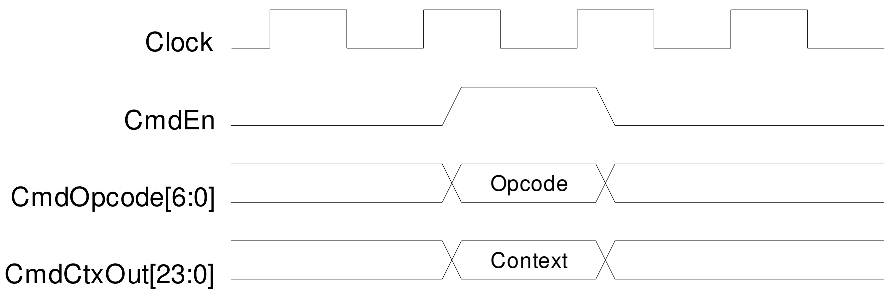

Figure 3.4: Command interface timing diagram.

3.4 Data Block

This block is part of the interface which allows the RCE to receive data from the FEB. Table 3.4 shows the interface signals between the RCDI data block and the user data block.

The start of transmission of a frame is signaled by the user logic by asserting `FrameTxSof`. An assertion of `FrameTxSof` must always be followed by an assertion of `FrameTxEof` before `FrameTxSof` may be asserted again. The exchanged data word `FrameTxData` is 16 bits wide. One data word is delivered from the user logic to the data block when `FrameTxEnable` is high for any given clock tick. The signal `FrameTxDataWidth` is always one during data transfer to indicate the exchanged word is two bytes wide, but it may be zero at the end of the frame if the last word is one byte wide.

Figure 3.5 shows the timing diagram used to send a frame on the data interface.

Table 3.4: Data Interface. Direction is from the RCDI core point of view.

Signal	Width	Direction	Description
`FrameTxEnable`	1	Input	Signal from user logic to deliver one data word to data block
`FrameTxSof`	1	Input	Asserted by user logic to indicate data word is the first of a frame
`FrameTxDataWidth`	1	Input	Indicate width of current data word
`FrameTxEof`	1	Input	Asserted by user logic to indicate data word is last of the frame
`FrameTxEofe`	1	Input	Asserted with `FrameTxEof` when user logic wishes to indicate an error exists within the passed frame data
`FrameTxData[15:0]`	16	Input	Data word provided by user logic for transmission

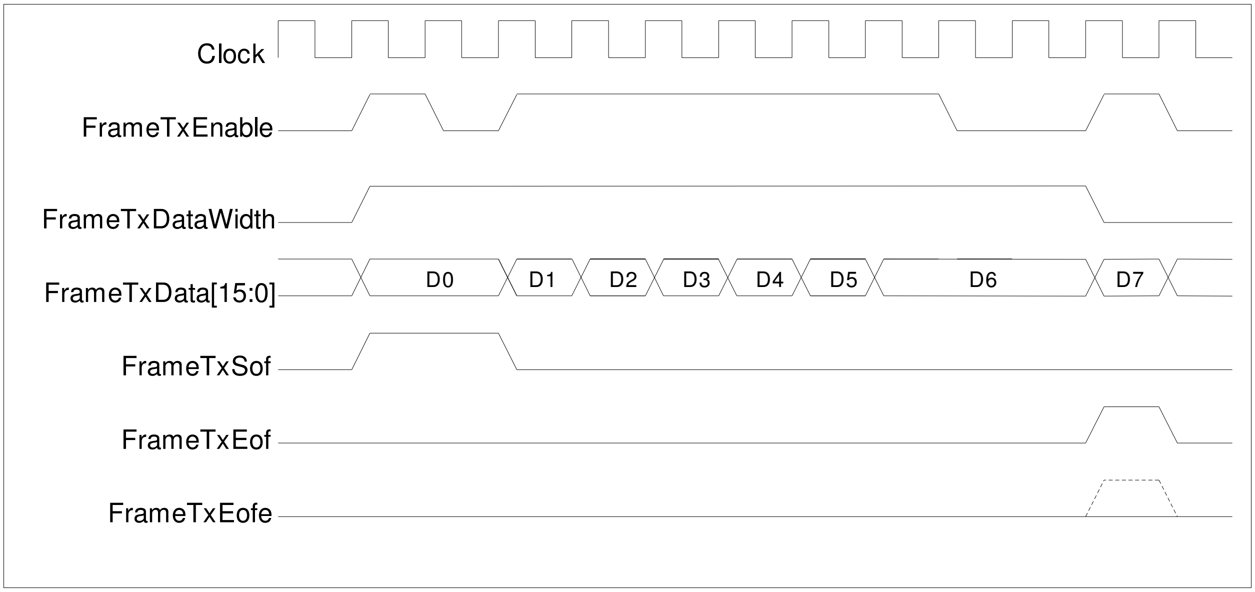

Figure 3.5: Data interface timing diagram.

Chapter 4: Software Interface

4.1 Register Interface

The register interface will allow the RCE to access 32 bit registers using a 24 bit address space through the `Reg` class:

```
namespace RceRcd {
  class Reg {
  public:
    unsigned write(unsigned addr, unsigned value) throw (Error);
    unsigned set(unsigned addr, unsigned mask) throw (Error);
    unsigned clr(unsigned addr, unsigned mask) throw (Error);

  public:
    unsigned read(unsigned addr) throw (Error);
  };
};
```

Only the 24 least significant bits of `addr` will be used to identify the register. The `read` method returns the value of the register specified by `addr`. The `set` method sets to high the bits specified by mask. The `clr` method sets to low the bits specified by mask. The `set`, `clr` and `write` methods all return the new register value.

The exception class `Error` is declared as:

```
namespace RceRcd {
  enum Reason {Timeout, InvalidAddress};
  class Error {
  public:
    Error(Reason, const char* msgformat, ...);

  public:
    const Reason reason;
  };
};
```

The `Timeout` entry in the `Reason` enumeration indicates that the FEB failed to acknowledge the command in a specified interval of time. The `InvalidAddress` entry indicates that the requested register address is out of the range of validity for that FEB.

4.2 Command Interface

The command interface will allow the RCE to send commands to the FEB through an out-of-band channel using the `Cmd` class:

```
namespace RceRcd {
  class Cmd {
  public:
    void send(unsigned char opcode, unsigned context);
  };
};
```

The meaning and the possible values for `opcode` depend on the specific remote platform. Only the 24 least significant bits of `context` and the 7 least significant bits of `opcode` are meaningful. The context word is opaque to the FEB which will echo it back to the RCE for all those commands which require a response. The context word will be used by the RCE to associate a response from the FEB with a given command.

Chapter 5: Packet Format

This section is not required to specify the interface, but it's here for completeness. Note that the packet format might change and still leave unmodified the interface described in the previous sections.

The byte order in the exchanged packets will be big-endian.

5.1 Command Messages

All the command packets issued by the RCE to the FEB are made of four long words for a total of 16 bytes. See figure 5.1. The first word of these packets contains the 24 bits of the opaque context number, 7 bits for the opcode and one bit to indicate internal register access. If this bit is asserted, the message is processed by the register block and is not forwarded to the user logic. The other words are currently reserved and must be zero.

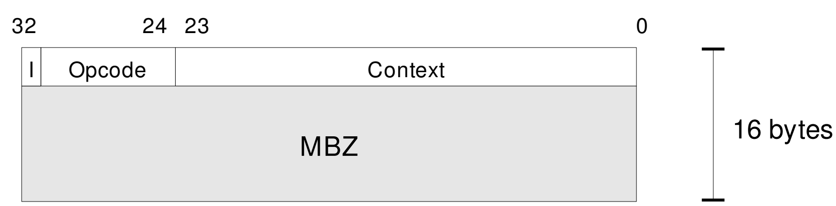

Figure 5.1: Command message.

The reply packets, issued by the FEB to the RCE using the data interface, have a fixed size header of 16 bytes and they may have an optional payload. See figure 5.2. Since the payload depends on the opcode and on the platform, its format cannot be defined in this document and it must be described in the platform specific documentation. The first word in the header is opaque to the FEB and it simply echoes the first word of the request message. The other words are currently reserved and must be zero.

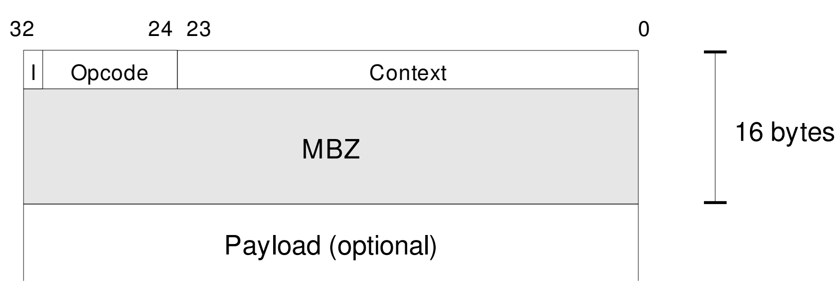

Figure 5.2: Command message reply.

5.2 Register Access Messages

All the register access packets sent by the RCE to the FEB are made of four long words for a total of 16 bytes. See figure 5.3. The first word of these packets contains 24 bits for the address, two bits for the operation code and one bit to indicate internal register access. If this bit is asserted, the message is processed by the register block and is not forwarded to the user logic.

The second word in the packet contains the register value. This field is used only for write operations and must be zero for read operations. The other words are currently reserved and must be zero. Valid values for the operation field are shown in table 5.1.

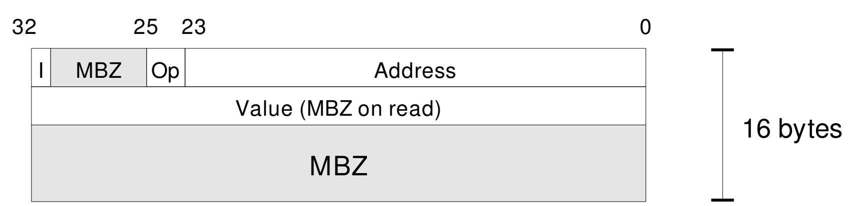

Figure 5.3: Register access message.

Table 5.1: Valid register access operation codes.

Name	Value	Description
READ	0x0	Invoked by the `read()` method
WRITE	0x1	Invoked by the `write()` method
SET	0x2	Invoked by the `set()` method
CLEAR	0x3	Invoked by the `clr()` method

The reply packets, issued by the FEB to the RCE in response to register access requests, have a fixed size header of 16 bytes. See figure 5.4. The first word in the header echoes the address, the operation code and the internal bit from the request message. It also includes a failed bit (F) to indicate that the user logic detected and invalid address and a timeout bit (T) to indicate that the user logic didn't acknowledge the operation within the timeout limit.

The second word indicates the value of the register for read operations and the value written to the register for write operations. The last two words are currently reserved and must be zero.

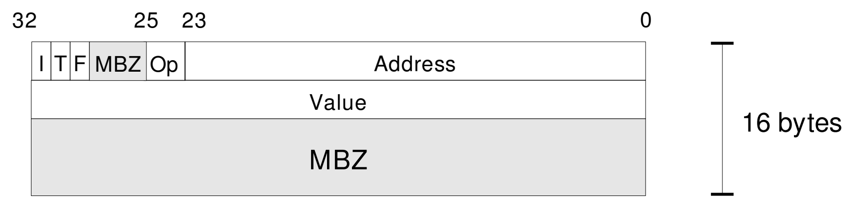

Figure 5.4: Register access message reply.

Bibliography

[RCE07] Mike Huffer, *The Reconfigurable Cluster Element (to be written)*, 2007

[PGP07] Mike Huffer, Ryan Herbst, *The Pretty Good Protocol (to be written)*, 2007

[VX406] Xilinx, *Virtex-4 Family Overview*, 2006

[MGT06] Xilinx, *Virtex-4 RocketIO Multi-Gigabit Transceiver User Guide*, 2006