# RDBE Host Software

Doc No: X3C 2009_07_21_1
TODO: Add appropriate document number

**Document history**

| Change date | Changed by | Version | Notes |
|---|---|---|---|
| 09-07-21 09:12 | Mikael Taveniku | PA1 | New document |
| 09/08/27 | Mikael Taveniku | PA2 | Included DBE Commands |
| | | | Added RDBE_server implementation description |

# Table of Contents


1OVERVIEW....................................................................3
1.1GENERAL....................................................................3
1.1.1U-BOOT....................................................................4
1.1.2LINUX....................................................................4
1.1.3RDBE SERVER....................................................................5

2RDBE DEVICE DRIVER....................................................................5
2.1EXTERNAL INTERFACE....................................................................6
2.2INTERNAL DESIGN....................................................................7

3RDBE CLIENT....................................................................7
3.1TO DO....................................................................7
3.2IMPLEMENTATION....................................................................7

4SYMBOLIC NAME DEFINITION (TBD)....................................................................7
4.1MACRO DEFINITION FILE FORMAT....................................................................8

5RDBE SERVER....................................................................8
5.1USAGE....................................................................8
5.2IMPLEMENTATION....................................................................8
5.2.1GENERAL OVERVIEW....................................................................9
5.2.2CONNECTION LISTENER....................................................................9
5.2.3DEVICE ACCESS....................................................................9
5.2.4CONNECTION HANDLER....................................................................9
5.2.5COMMAND INTERPRETER AND EXECUTION....................................................................9
5.2.6COMMANDS....................................................................10
5.2.7GENERAL SERVICES....................................................................11

6RDBE SERVER VSI-S LOW-LEVEL COMMANDS....................................................................11

7RDBE SERVER VSI-S BASE COMMAND SET....................................................................12

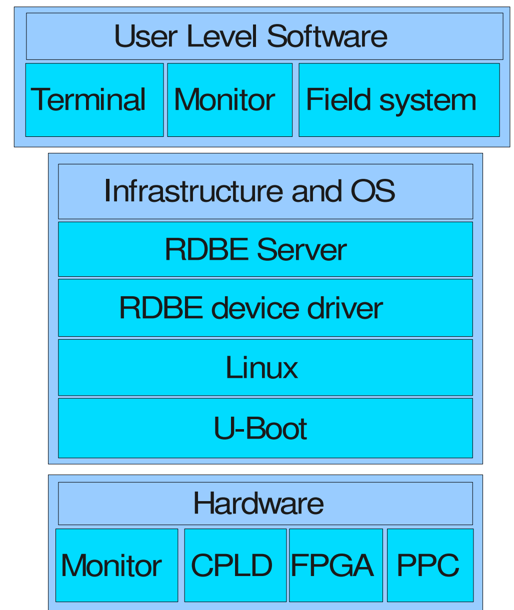
8HIGH LEVEL SOFTWARE NOTES....................................................................13
8.1U-BOOT....................................................................13
8.2LINUX CONFIGURATION....................................................................13

# 1 OVERVIEW

This document describes the RDBE software running on the Roach board.
The structure consist, from the layer closest to the hardware and up of:

- U-Boot: The boot loader for the system. This software is responsible to setup the hardware mapping of the device to processor addresses. Specifically the PowerPC EBC registers, as well as to provide a boot platform for the LINUX operating system.
- Linux Kernel and a  Debian (Etch) root file system: This is the base operating system for the board.
- RDBE Device driver (RDBE_dev): The  RDBE device driver provides read and write access to the FPGA and CPLD devices on the Roach board. This driver is a single client device and is designed to work with the RDBE server software.
- RDBE Server: The RDBE server provides a multi-user TCP based interface to the RDBE board. It understands a VSI-S based and a command line based command set.
- RDBE Client: A command line based interface to communicate with the RDBE Server software. This interface is designed to be a monitor program for the FPGA designer enabling easy access to facilities on the board.
- VSI-S Compatible system software (outside scope of this document):.
- RDBE Monitor: The RDBE monitor software provide continuous monitoring of the RDBE board. (TO BE DEFINED)

- U-Boot
  - Provides basic hardware setup
  - Boot facilities for Linux
  - Simple command line monitor
- Linux
  - Unix user and software facilities
  - Based on Debian / Etch distribution
- RDBE Device driver
  - Provide read, write access to RDBE hardware
  - Provision hardware access
  - Programming functions for FPGA
- RDBE Server
  - User level service that provides TCP based access to RDBE hardware through the RDBE device driver
  - Understands VSI-S commands
  - Extended command set for debug and FPGA development
- RDBE Monitor
  - Application providing monitoring and control of the RDBE board and Application.




In addition to the base software modules mentioned above there is a Macro facility enabling the FPGA designer to use symbolic constants for register addresses and operations on the board.

## 1.1 GENERAL

The RDBE haystack software is built to provide a straight forward, easily maintainable software stack that will be flexible enough to support the applications planned.

General:

- The application software shall be small enough to fit into an embedded setting

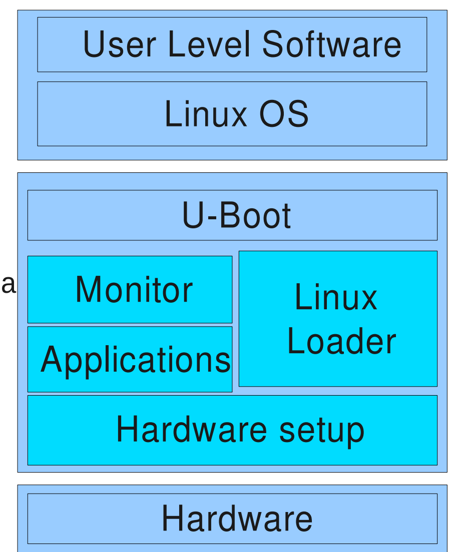
- System shall have the option to boot completely by itself, this means that all applicable software shall fit into the on-board Flash.
- There shall be an option to run the board in “full Linux mode” using an external NFS host. In this case the root file system will reside on the host machine and be mounted over the network.

### 1.1.1 U-Boot

The U-boot module provides the base software of the board that will bring  it up, set the device mappings and provide the boot environment for Linux. In addition to the boot layer for Linux it also has an interactive mode suitable for debug of the board, see u-boot documentation.

The modifications to U-boot for RDBE is kept to a minimum and follows the Sequoya board BSP to a high degree. The specific modifications for the RDBE device mapping is contained to a few files in the hs_uboot project (see the roach_main document and the u-boot manual) .

- U-Boot
  - Is a boot loader designed to be portable to many different architectures and to provide a boot environment for Linux
- Standalone Applications
  - U-boot allows for standalone applications to run in the u-boot environment, even without an operating system.
  - Programs run in supervisor mode and the entire machine is availa
- Haystack Modifications
  - Memory map modified to accommodate new applications
  - Power-PC IO bus setup modified for FPGA access
  - Standalone application for test of Ethernet and FPGA
  - Boot setup developed for RDBE software stack




The most important part of u-boot for Roach is to setup the hardware, memory maps etc. and to provide a basic IO system so that Linux can be booted. It also has a monitor interface, which is easily extended with user commands.
There is also a facility to create and run user applications in a “c”-like environment directly under u-boot or as part of the u-boot startup phase.

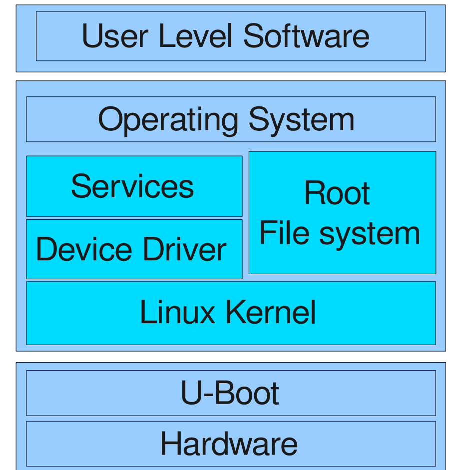
### 1.1.2 Linux

The Linux port is based on the work out of South Africa, which in turn is based on the Sequoya reference BSP.
In addition to the necessary changes to support the RDBE board, the hs_linux project contains the kernel mode device driver for RDBE.
The necessary configuration parameters are contained in the modified configuration file and the kernel driver itself. In all other aspects this is an off the shelf Linux kernel.

In order for the operating system to work, it needs a ROOT file system. A root file system essentially provides the initial mount point for Linux, as well as all the initialization files, command interpreter etc. that enables the system to interact with the world.

Currently the software uses a Debian Etch distribution in two variants, one that contain a complete Debian distribution used for the FULL development environment, and a bare bones system that can fit into the on-board FLASH. The later system provide the minimum of services in order to run the rdbe_server and monitor applications.

- Linux (2.6.x)
  - Off the shelf kernel image, modified with BSP for Roach
- RDBE Device Driver
  - Provide fast access to RDBE hardware
  - FPGA, CPLD, SMAP, Monitor, Program facility for FPGA
- Services Layer
  - Normal Linux services available
- Root File System (Debian Etch)
  - Initial Kernel mount point
  - Provide startup scripts and programs for kernel operation




The Kernel module device driver provides basic read/write access to the underlying hardware devices on the board, currently FPGA, SMAP, and CPLD, (Roach Monitor and FLASH is planned). Software can access the hardware through the “rdbe_dev” device.

### 1.1.3 RDBE Server

The RDBE Server software is an application that provide applications access to the RDBE hardware and other facilities. It is the intended connection point for any software that wants to access hardware resources on the RDBE.
It sits on top of the RDBE device driver and ensures sequential and provisioned access to the driver for applications.

As an external interface the server listens for TCP traffic internal or external to the RDBE. When a connection request comes in it provides a service thread for the requester that understands how to service the request.

The server can service a configurable amount of simultaneous connections while ensuring hardware protection as well as sequential access for all requesters. It is possible to provide weighted scheduling or prioritized access, however the current implementation orders accesses in time order and limits each requester to one outstanding request each.
This scheme essentially provides a round robin schedule for hardware access as well as fairness.

The server application has a VSI-S compatible command parser as default. It is possible to allow other command protocols.

The commands implemented for the RDBE is defined in the command module of the server application. This module is designed to be easily extensible for new commands or new hardware personalities in the FPGA or on-board application.

# 2 RDBE DEVICE DRIVER

The RDBE device driver provides a read, write and “mmap” interface to the devices on the roach board.

The implementation files resides in the “hs_roach/kernel/haystack” directory in the hs_roach project (Linux kernel for the RDBE board port).
This device driver is implemented either as a loadable or built in kernel module.
The Linux Makefiles are modified so that there is a configuration option for including the kernel driver when Linux is built. (See software Notes)

RDBE_dev, which is the device name of the driver in the running system is designed to be as simple as possible and to move as much of the application layer work to a user mode application. With this scheme most programming can be done outside the kernel and easier modified and debugged.

## 2.1 EXTERNAL INTERFACE

The external interface to the RDBE_dev id defined below. It basically is a Read/Write and IOCTL interface. The Read and Write functions provide (a seek) random access to all device memory mapped to the device driver, but may be somewhat slow to use.
The MMAP call is currently implemented but not supported due to hardware issues (TBD). The preferred access to the hardware is through the IOCTL methods that gives direct access to the hardware, through the RDBE_server software module.

| Name | Description | Comment |
|---|---|---|
| Read | Read data from the roach device as a stream of bytes. | Devices are mapped into memory in sequence: FPGA, SMAP, CPLD |
| Write | Write data to the devices | The number of bytes and the alignment must be correct for the device written. |
| IOCTL | General IO | |
| - RDBE_FPGA_READ | Read data from FPGA | Non aligned accesses are not supported |
| - RDBE_SMAP_READ | Read data from SMAP | Non aligned accesses are not supported |
| - RDBE_CPLD_READ | Read data from CPLD | Non aligned accesses are not supported |
| - RDBE_FPGA_WRITE | Write data to FPGA | Non aligned accesses are not supported |
| - RDBE_SMAP_WRITE | Write data to SMAP | Non aligned accesses are not supported |
| - RDBE_CPLD_WRITE | Write data to CPLD | Non aligned accesses are not supported |
| - RBDE_DEV_RESET | Reset the FPGA | Non aligned accesses are not supported |
| - RDBE_DEV_CONFIGURE | Program the FPGA from memory | |
| Open | | |
| Release | | |

The Read / Write standard device access interface simulates a single device with sequential access. The devices are mapped into the virtual file in sequence. The FPGA memory space starts at offset 0 and and has a window size of 128MByte, then the SMAP register space is mapped into the file at RDBE_SMAP_OFFSET

- RDBE_FPGA_OFFSET = 0x0000 0000
- RDBE_SMAP_OFFSET = 0x0800 0000
- RDBE_CPLD_OFFSET = 0x0810 0000

There are two additional data types that are defined for the device driver:

- rdbe_configure_cmd_t: That contain the length of the configuration file buffer and a pointer to the buffer itself.
- rdbe_rdwr_cmd_t: This structure contain information for a read or write IOCTL command.
- access_type: 1,2 or 4 byte access.

- offset: start address from device base address.
- length: The number of items of access type
- buffer: Pointer to a buffer that hold data to be written, or big enough for data returned from a read operation

The driver is written for performance and minimal checking is done inside the code. This means that great care should be taken to ensure that parameters are correct and that memory buffers are properly allocated by the caller.

### 2.2 INTERNAL DESIGN

RDBE (TBD)

## 3 RDBE CLIENT

The terminal program provides a simple text based user interface to the RDBE device hardware.
The terminal program is line based and expects each command to end with a new-line. Parameters are separated by “:”.
Numbers are allowed in binary, octal, decimal and hex format. When a number is specified base specifier are required (0x , b or o postfix for binary or octal numbers and none for decimal).
The terminal program understands the following commands:

| Command | Parameters | Description | Comment |
|---|---|---|---|
| Low-Level | | See RDBE Server Low Level Commands | |
| Base | | See RDBE Base Command Set | |

### 3.1 TO DO

Add methods to write more than one word at the time for the “wr” commands. The parser should be able to count on the fact that the data is correct and contained on a single line?
xxxwr {b|w|l} offset {data}* ';' where the parser makes best effort to interpret the data in the string until ';' is found.

### 3.2 IMPLEMENTATION

Currently the RDBE_CLIENT is implemented by the “rdbe_tcp_client” executable from the hs_server project.
The “rdbe_tcp_client” takes the server IP address and the server port number as parameters.
Example: ./rdbe_tcp_client 192.168.0.123 5000
to connect to a RDBE Server at IP address 192.168.0.123 listening on (tcp) port 5000

## 4 SYMBOLIC NAME DEFINITION (TBD)

// This is not implemented yet !! //

In addition to the basic commands each personality (FPGA program for example) has a natural way of describing registers with names instead of physical addresses. Furthermore the registers interpretation may be described by symbolic names rather than a binary representation.

The RDBE server / term can load a register definition file that define symbolic names to register values. It can also be used to define symbolic names for values to set registers to.

The command to load a register definition file is loaddef <filename>

## 4.1 MACRO DEFINITION FILE FORMAT

The definition file is fixed format and line based, all definitions end with a new line. A register definition is defined as follows:

name ::= letter+ non-delimiter-character*
number ::= (decimal_num | octal_num | hexadecimal_num | binary_num)
register_definition ::= 'REGISTER' name width access type address
value_definition ::= 'VALUE' name number
comment ::= '#' []*
access_type ::= ('r' | 'w' | 'rw')

These definitions can then be used in place of numbers in the terminal program.
Example:
REGISTER adc_0, 16, rw, 0x400
VALUE adc_gain_min, 000b
# this is a comment

# 5 RDBE SERVER

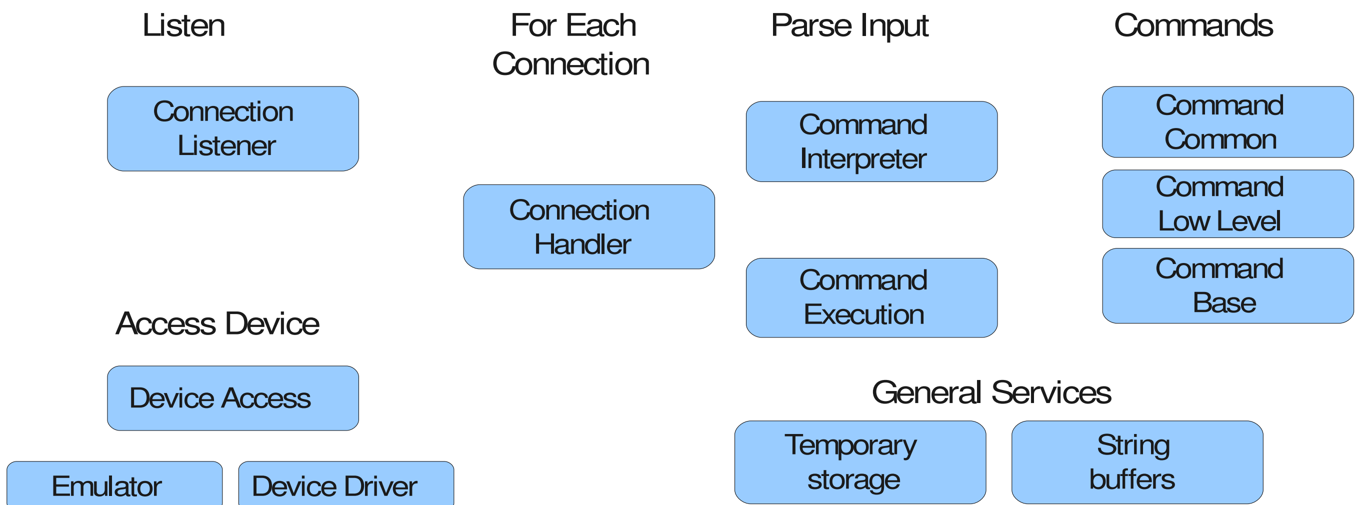
The RDBE server is the application that implements the external access to the RDBE board.

## 5.1 USAGE

Currently the RDBE server is implemented by the executable “rdbe_server” the application is run as rdbe_server <port number> <max connections>
for example “./rdbe_server 5000 5” starts the rdbe_server and instructs it to listen on (tcp) port 5000 and to allow a maximum of 5 simultaneous connections.

## 5.2 IMPLEMENTATION

The RDBE server is implemented in the rdbe_server project on the vault SVN server.
Essentially it can be roughly be described as in the figure below. There is a connection listener that listen for incoming connections, a connection handler for each connection, a parser that interprets incoming commands and requests and commands that can be executed. In addition there is a facility for device access and internal services.



### 5.2.1 General overview

The project consist of the following implementation files

- server_main: implements the main entry point to the program as well as the device access and connection handler functions
- cmd_parser: Implements the parser and command execution functions. Today the parser consist of two main parts, one that implements parsing of text to commands, and one that keeps a list of available commands to the user.
- cmd_common: Implements a set of legacy commands that are currently not in use
- cmd_low_level: Implementation of low-level access commands used for debug purposes
- cmd_base: Implements VSI-S RDBE commands used for application access to the RDBE
- rdbe_emulate: implements a STUB that can be used to provide responses to the application in while debugging without hardware in the loop
- serv_storage: Implements a thread (safe) based storage buffer for the application
- Makefile and Makefile.native is the makefiles for the project.

The application is based on GNU-C and Pthreads, no special libraries are used. The application runs on both x86 and PPC platforms. Note however, that the application today is not endian-safe.

### 5.2.2 Connection Listener

The connection listener listens for connection requests on the specified TCP ports. If a valid request comes in and there are available connections (less than max sessions are in use) it creates a new connection handler thread for the connection. The code for the connection listener is implemented in server_main.

### 5.2.3 Device Access

Device access is simply a wrapper around the rdbe_dev IOCTL command with a MUTEX guarding access to to the devices. This code is implemented in server_main and called from the command functions needing access to the roach hardware.

In addition to the device access through the rdbe device driver there is a emulation stub that is implemented in the emulate.c file. This file simply provides an “emulated” rdbe_iotl function so that the server software have something to call when real hardware is not available. The emulator is there to simplify hardware and software debug.

### 5.2.4 Connection handler

The connection handler is the main function for each connection. It reads commands and requests from the input stream, passes each “line” to the command interpreter/parser that if successful returns a executable command packet to the handler. If so, the handler passes the packet to the command execution.
When parsing and/or execution is done the handler returns the results to the requester and goes back to look for a new command.
The connection handler is implemented in server_main.

### 5.2.5 Command Interpreter and Execution

The command interpreter implements functions to parse a command line compatible with the VSI-S. It also holds a list of valid commands, those that are enabled and implemented in the cmd_common, cmd_low_level, and cmd_base. When called the parser builds a potential command based on the input buffer. If this process is successful (meaning that the command line was a well formed VSI-S sentence) it then scans the list of implemented commands looking for one that matches the input.

If such command is found it then builds a valid (if possible) argument list for the command and returns an executable command to the connection handler.

An executable command consists of a pointer to the function implementing the command and the argument lists to the command.

The functions implementing this functionality is contained in the parser.c file.

### 5.2.6 Commands

The system is built so that it is should be straight forward and relatively easy to implement new commands as well as keeping separation between parsing commands and implementing commands.
When commands are called from the server application the implementation of the command knows that the argument list is well formed.
However, VSI-S allows variable length argument lists for all commands as well as optional arguments to some commands. It is further complicated by the fact that omission of an argument sometimes means “don't change the parameter”, sometimes “use default value” and other times it is invalid. In our current implementation of the command interpreter and the description of the commands we leave the decision to the actual implementation of command to decide what to do with “null” valued arguments.

Each command consist of two parts on function that implements the command itself, does the actual work, and one function that describes the command to the system.
In the current implementation these pairs are implemented as “VSIS_FUNCTION()” and “insert_vsis_function()”

Each function looks like this:

```
**
 * Read a number of bytes from the FPGA device
 * @param argc    - should be 3 <access_type> <offset> <number of byte>
 * @param argv    - pointers to the parameters
 * @param retVal - buffer with the data (allocated here)
 * @param retLen - length of buffer in BYTES
 * @return        - 0 on success
 */
int vsis_fpga_rd_cmd(int argc, void **argv, void **retVal, int *retLen)
{ …  }
```

The arguments to the function is the argument list built by the parser and the return values are contained in the retVal and retLen pointer that each function is responsible for allocating memory for.
The function returns text strings in a valid VSI-S format that is sent to the client.

The insert_XXX_XXX function looks like this:

```
/**
 * Insert the fpga read command into command queue.
 */
void insert_vsis_fpga_rd(void)
{
            char *name = "fpgard";
            char *usage= "fpgard=<access type>:<offset>:<length>;";
            int nParams = 3;
            int cmdType = CMD_VSIS_COMMAND;
            int paramTypes[3]={CMD_PARAM_ACCESSTYPE,CMD_PARAM_UNSIGNEDLONG,
CMD_PARAM_UNSIGNEDLONG};
            cmd_insert_cmd_desc(name,usage,nParams,paramTypes,vsis_fpga_rd_cmd,cmdType);
}
```

In the code snippet above, the fpgard command is defined with name=fpgard, a hint on usage in the usage string, and a description of the arguments required of the function as well as a definition of what type of function it is. The last line enters the function description into the global list of available commands.

Command Common: Implements basic functionality for the command implementations. The commands implemented in this file are currently not used.
Command Low-Level is described in detail in next chapter. It implements basic low level device access commands.
Command Base: Implements VSI-S commands needed for the application level commands. These commands are described under heading 7.

### 5.2.7 General Services

There are a few extra services implemented in string_buffer and serv_storage that provide a dynamic string buffer facility and a buffer storage facility.

# 6 RDBE SERVER VSI-S LOW-LEVEL COMMANDS

In order to access the hardware on the Roach board, there is a set of low level commands  directly accessing hardware on the board. These commands are intended for manual debug of the board or direct access to hardware resources.

Theses commands are VSI-S syntax compatible, but are not intended to be included in standard software definitions.

| Command | Parameters | Description | Comment |
|---|---|---|---|
| exit | | End program | |
| fpgard | {b,w,l} offset, length | Read length data from FPGA starting at offset | |
| fgpawr | {b,w,l} offset, datum | Write one word of data to FPGA | |
| cpldrd | {b,w,l} offset, length | Read length data from CPLD starting at offset | |
| cpldwr | {b,w,l} offset, datum | Write datum to CPLD at offset | |
| smaprd | {b,w,l} offset, length | Read length data from SMAP starting at offset | |
| smapwr | {b,w,l} offset, datum | Write a word of data to SMAP | |
| fpgaprg | Filename | Programs FPGA with local file “filename” | |
| loaddef | filename | Load register definitions from local file | |
| Loadbuf | BUF <0..9> num_byte [bytes] | Load bytes from client to temporary storage on server | |
| fpgaprgbuf | BUF <0..9> | Program the FPGA with contents of buffer N | |

Notes:
For the latest command definitions and details see rdbe_developer_interface document.

# 7 RDBE SERVER VSI-S BASE COMMAND SET

The DBE version 2 system has functionality that was once located in the Mark5 unit but that has now been moved to the generating source. The specified command set reflects this utilizing commands that were previously issued to the Mark5B system. In addition newer commands are required to add support for features that previously did not exist for communicating to the Mark5C recording system.
The command set will be presented as set of commands and queries separated into 4 main categories: Initialization, Timing, General and Data Mode, that were presented in the DBE Software Command Requirement Document Memo.
From the external application perspective, the RDBE software recognize the following commands:

| Command | Description | Comment |
|---|---|---|
| rdbe_personality | set / get the DBE FPGA bit code personality | |
| mode | set / get data transmission mode | |
| rdbe_clock_set | set – set the clock parameters | |
| rdbe_DOT | Get the Data Observable Time (DOT) clock information | |
| rdbe_DOT_set | set the DOT clock on next 1pps tic | |
| rdbe_DOT_inc | Increment the DOT clock | |
| status | get system status (query only) | |
| rdbe_sw_version | get the software version information from the DAS | |
| rdbe_ifconfig | set / get DBE 10G network interface configuration | |
| rdbe_tid2addr | set / get the channel ID association resolution to either IP or MAC address | |
| rdbe_arp | set / get the IP to MAC address resolution | |
| packet | set / get packet transmission criteria | |
| data_xfer | send a valid data stream out of the DBE ON/OFF | |
| data_format | Set the packet format mode to either the Mark5C native mode or Mark5B compatibility mode | |
| quantize_seed | set / get the seed for the gain settings | |
| quantize_<chid> | Get channel quantization values - Query | |

For more details and the latest revision of the command set, see rdbe_external_interface document.

# 8 HIGH LEVEL SOFTWARE NOTES

## 8.1 U-BOOT

Boot command:
-

## 8.2 LINUX CONFIGURATION

/etc/init.d
Install device driver as a module
make sure to remove bof-parts from driver
resolve password change policy